Miniprojeto de Inteligência Artificial

Detecção de Comportamento de Direção Anormal com Machine Learning e Deep Learning

Curso: Ciência da Computação — 5º Período
Instituição: UNICAP – Universidade Católica de Pernambuco
Disciplina: Inteligência Artificial
Professor: Francisco Madeiro
Aluna: Maria Clara Soares
Semestre: 2025.1

---

1. Introdução

Acidentes de trânsito associados a comportamentos de direção agressiva, sonolenta ou desatenta ainda figuram entre as maiores causas de mortalidade no trânsito mundial. Diante desse cenário, este projeto visa aplicar técnicas de Inteligência Artificial para detectar tais comportamentos com base em dados reais, por meio da replicação do artigo “Abnormal Driving Behavior Detection: A Machine and Deep Learning Based Hybrid Model” (Uddin et al., 2025).

Utilizando o conjunto de dados UAH-DriveSet, implementei uma abordagem híbrida com algoritmos de Machine Learning (ML) aplicados a dados sensoriais e modelos de Deep Learning (DL) aplicados a imagens de direção. A proposta buscou explorar a capacidade preditiva dessas técnicas, respeitando as limitações técnicas reais do ambiente de desenvolvimento.

---

2. Artigo Base

Título: Abnormal Driving Behavior Detection: A Machine and Deep Learning Based Hybrid Model
Autores: Md. Ashraf Uddin et al.
Periódico: International Journal of Intelligent Transportation Systems Research
Ano: 2025
DOI: https://doi.org/10.1007/s13177-025-00471-2

---

3. Objetivo do Projeto

- Reproduzir a proposta híbrida do artigo combinando ML e DL na detecção de comportamentos de direção.
- Avaliar a performance dos modelos sob condições computacionais reais.
- Adaptar estratégias técnicas para garantir comparações justas com a CNN base.
- Documentar cada etapa com rigor técnico, destacando as semelhanças e divergências em relação ao artigo original.

---

4. Dataset

UAH-DriveSet é um conjunto de dados público que reúne amostras de direção real realizadas por 6 motoristas em rodovias e zonas urbanas, sob três tipos de comportamento:

- Normal
- Agressivo
- Cansado

O dataset possui duas fontes principais:

- **Dados tabulares:** velocidade, aceleração, frenagem, ângulo de direção, tipo de estrada, entre outros.
- **Imagens:** frames extraídos manualmente dos vídeos da base.

Para Deep Learning, os frames foram organizados em pastas por classe. As imagens foram redimensionadas para 160×160 na CNN Base e para 224×224 na ResNet50, respeitando os requisitos de entrada específicos de cada arquitetura.

---

5. Tecnologias Utilizadas

Linguagem: Python 3.10.11
Ambiente: Jupyter Notebook no VS Code, executado localmente em CPU (Intel i3, 8GB RAM)

Bibliotecas:

- **Manipulação e Pré-processamento:** pandas, numpy, os, shutil, glob, sklearn.preprocessing.StandardScaler, RFE, ImageDataGenerator, BalancedImageGenerator
- **Modelagem ML:** RandomForestClassifier, SVC, KNeighborsClassifier, GridSearchCV
- **Modelagem DL:** tensorflow.keras.models.Sequential, ResNet50, EfficientNetB6 (tentativa abortada), Conv2D, MaxPooling2D, Flatten, Dropout, BatchNormalization, Dense, GlobalAveragePooling2D
- **Treinamento DL:** Adam, categorical_crossentropy, EarlyStopping
- **Avaliação:** accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
- **Visualização:** matplotlib, seaborn

---

6. Modelos Implementados

6.1 Machine Learning

- **Random Forest**
- **Gradient Boosting**
- **KNN**
- **SVM (RBF)**
- **Decision Tree**
- **Logistic Regression**

Os três primeiros foram selecionados como os de melhor, médio e pior desempenho para análise comparativa com o artigo, mas todos os seis foram implementados e avaliados.

6.2 Deep Learning

- **CNN Base:** construída do zero com 3 blocos de convolução + pooling + dropout + dense + softmax.
- **ResNet50:** versão pré-treinada com include_top=False, camadas convolucionais congeladas e topo com GAP + Dense(128) + Dropout(0.4).
- **EfficientNetB6:** planejada, mas não foi possível treinar devido à exigência computacional extrema (detalhes abaixo).

---

7. Arquitetura CNN Original vs CNN Implementada e Estratégias de Comparação Justa

CNN no Artigo Original

A CNN foi construída com cinco blocos convolucionais, seguidos de camadas densas com Dropout e BatchNormalization. A entrada eram imagens da base SFD3, e o modelo foi treinado por 20 épocas com lotes balanceados e GPU. A função de ativação foi ReLU nas camadas internas e softmax na saída, com Adam como otimizador.

Modificações na Nossa CNN Base

Desenvolvi uma versão mais enxuta da CNN, com:

- Redução para 3 blocos convolucionais
- Dropout(0.5) como regularizador
- Menor profundidade de filtros
- Imagens redimensionadas para 160x160
- Topo: Flatten → Dense(128) → Dropout → Softmax

Estratégias de Comparação Justa com a ResNet50

- Topo idêntico em ambas as redes
- Mesma divisão de dados (70/30 com `seed`)
- Mesmo aumento de dados (`ImageDataGenerator`)

- Mesmo balanceamento de batches
- Avaliação com as mesmas métricas

---

8. Limitações Computacionais e Adaptações

Para viabilizar a ResNet50 em CPU:

- Congelei as camadas convolucionais
- Redimensionei imagens para 224×224
- Ajustei batch_size=24, steps_per_epoch=250
- Configurei multiprocessing
- Apliquei o mesmo topo da CNN Base

Mesmo assim, **cada época levou cerca de 2 horas**. Durante a **segunda época**, o sistema travou, impossibilitando a continuidade. Avaliei os resultados com base na **primeira época**.

---

9. Avaliação dos Modelos de Machine Learning

Interpretação das Células 2 a 7

- **Célula 2–5:** verificam a estrutura dos arquivos, leitura dos dados sensoriais (acelerômetros e GPS), extração de segmentos e identificação das classes a partir dos nomes das pastas.

```
RAW_ACCELEROMETERS encontrados: 40
RAW_GPS encontrados: 40

Acelerômetro 1: D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151110175712-16km-D1-NORMAL1-SECONDARY\RAW_ACCELEROMETERS.txt
Existe? True
Primeiras linhas:
  6.94 1 0.017 -0.011 0.018 -0.005 0.008 0.018 -1.523 0.015 0.012
  7.03 1 0.046 0.007 0.019 0.016 -0.002 0.018 -1.522 0.012 0.012
  7.14 1 0.052 -0.016 0.027 0.037 -0.005 0.018 -1.520 0.014 0.011
  7.24 1 0.015 -0.016 0.026 0.038 -0.009 0.024 -1.523 0.014 0.011
  7.34 1 -0.014 -0.017 0.040 0.012 -0.016 0.032 -1.525 0.012 0.011

Acelerômetro 2: D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151110180824-16km-D1-NORMAL2-SECONDARY\RAW_ACCELEROMETERS.txt
Existe? True
Primeiras linhas:
  0.59 0 0.004 -0.039 -0.013 0.000 -0.006 -0.002 -1.686 0.131 0.156
  0.59 0 0.001 -0.035 -0.011 0.001 -0.021 -0.007 -1.686 0.131 0.157
  0.59 0 -0.020 -0.017 -0.019 -0.006 -0.024 -0.012 -1.686 0.131 0.158
  0.59 0 -0.029 -0.010 -0.022 -0.020 -0.015 -0.018 -1.686 0.131 0.158
  0.59 0 -0.029 -0.010 -0.022 -0.026 -0.012 -0.021 -1.686 0.131 0.158

Acelerômetro 3: D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151111123124-25km-D1-NORMAL-MOTORWAY\RAW_ACCELEROMETERS.txt
Existe? True
Primeiras linhas:
  0.69 0 0.066 0.120 0.033 0.012 0.022 0.006 -1.487 -0.223 -0.373
  0.69 0 0.083 0.150 0.049 0.049 0.088 0.028 -1.652 -0.248 -0.415
  0.70 0 0.083 0.150 0.049 0.070 0.126 0.041 -1.652 -0.248 -0.415
  0.70 0 0.083 0.150 0.049 0.078 0.141 0.046 -1.652 -0.248 -0.415
  0.70 0 0.083 0.150 0.049 0.081 0.147 0.048 -1.652 -0.248 -0.415
```

Gerar + Código + Markdown


```
RAW_ACCELEROMETERS encontrados: 40
RAW_GPS encontrados: 40

Acelerômetro 1: D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151110175712-16km-D1-NORMAL1-SECONDARY\RAW_ACCELEROMETERS.txt
Existe? True
Primeiras linhas:
  6.94 1 0.017 -0.011 0.018 -0.005 0.008 0.018 -1.523 0.015 0.012
  7.03 1 0.046 0.007 0.019 0.016 -0.002 0.018 -1.522 0.012 0.012
  7.14 1 0.052 -0.016 0.027 0.037 -0.005 0.018 -1.520 0.014 0.011
  7.24 1 0.015 -0.016 0.026 0.038 -0.009 0.024 -1.523 0.014 0.011
  7.34 1 -0.014 -0.017 0.040 0.012 -0.016 0.032 -1.525 0.012 0.011

Acelerômetro 2: D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151110180824-16km-D1-NORMAL2-SECONDARY\RAW_ACCELEROMETERS.txt
Existe? True
Primeiras linhas:
  0.59 0 0.004 -0.039 -0.013 0.000 -0.006 -0.002 -1.686 0.131 0.156
  0.59 0 0.001 -0.035 -0.011 0.001 -0.021 -0.007 -1.686 0.131 0.157
  0.59 0 -0.020 -0.017 -0.019 -0.006 -0.024 -0.012 -1.686 0.131 0.158
  0.59 0 -0.029 -0.010 -0.022 -0.020 -0.015 -0.018 -1.686 0.131 0.158
  0.59 0 -0.029 -0.010 -0.022 -0.026 -0.012 -0.021 -1.686 0.131 0.158

Acelerômetro 3: D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151111123124-25km-D1-NORMAL-MOTORWAY\RAW_ACCELEROMETERS.txt
Existe? True
Primeiras linhas:
  0.69 0 0.066 0.120 0.033 0.012 0.022 0.006 -1.487 -0.223 -0.373
  0.69 0 0.083 0.150 0.049 0.049 0.088 0.028 -1.652 -0.248 -0.415
  0.70 0 0.083 0.150 0.049 0.070 0.126 0.041 -1.652 -0.248 -0.415
  0.70 0 0.083 0.150 0.049 0.078 0.141 0.046 -1.652 -0.248 -0.415
  0.70 0 0.083 0.150 0.049 0.081 0.147 0.048 -1.652 -0.248 -0.415
```


Gerar + Código + Markdown

```
' Pasta: D:\\UNICAP\\2025\\QUINTO PERIODO\\IA\\MINI PROJETO\\UAH-DRIVESET-v1\\D1\\20151110175712-16km-D1-NORMAL1-SECONDARY'

' Nome detectado: 20151110175712-16KM-D1-NORMAL1-SECONDARY'

 Rótulo: Normal
---

' Pasta: D:\\UNICAP\\2025\\QUINTO PERIODO\\IA\\MINI PROJETO\\UAH-DRIVESET-v1\\D1\\20151110180824-16km-D1-NORMAL2-SECONDARY'

' Nome detectado: 20151110180824-16KM-D1-NORMAL2-SECONDARY'

 Rótulo: Normal
---

' Pasta: D:\\UNICAP\\2025\\QUINTO PERIODO\\IA\\MINI PROJETO\\UAH-DRIVESET-v1\\D1\\20151111123124-25km-D1-NORMAL-MOTORWAY'

' Nome detectado: 20151111123124-25KM-D1-NORMAL-MOTORWAY'

 Rótulo: Normal
---

' Pasta: D:\\UNICAP\\2025\\QUINTO PERIODO\\IA\\MINI PROJETO\\UAH-DRIVESET-v1\\D1\\20151111125233-24km-D1-AGGRESSIVE-MOTORWAY'

' Nome detectado: 20151111125233-24KM-D1-AGGRESSIVE-MOTORWAY'

 Rótulo: Aggressive
---

' Pasta: D:\\UNICAP\\2025\\QUINTO PERIODO\\IA\\MINI PROJETO\\UAH-DRIVESET-v1\\D1\\20151111132348-25km-D1-DROWSY-MOTORWAY'

' Nome detectado: 20151111132348-25KM-D1-DROWSY-MOTORWAY'

 Rótulo: Tired
```

```python
# celula 5

# Caminho real de um arquivo com muitos dados
caminho_acc = r'D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151110175712-16km-D1-NORMAL1-SECONDARY\RAW_ACCELEROMETERS.txt'
caminho_gps = r'D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\D1\20151110175712-16km-D1-NORMAL1-SECONDARY\RAW_GPS.txt'

df_acc = pd.read_csv(caminho_acc, sep='\s+', header=None, usecols=[2,3,4], names=['ax', 'ay', 'az'])
df_gps = pd.read_csv(caminho_gps, sep='\s+', header=None, usecols=[0,1,2], names=['lat', 'lon', 'speed'])

min_len = min(len(df_acc), len(df_gps))
df = pd.concat([df_acc.iloc[:min_len], df_gps.iloc[:min_len]], axis=1)

print(" Total de linhas:", len(df))

window_size = 5
features_per_second = 10
janela_amostras = window_size * features_per_second
passo_amostras = 1 * features_per_second
total_janelas = 0

for i in range(0, len(df) - janela_amostras + 1, passo_amostras):
    total_janelas += 1

print(" Total de janelas possíveis:", total_janelas)
```

```
Total de linhas: 624
Total de janelas possíveis: 58
```

- **Célula 6:** contabiliza o total de linhas por gravação.

```
TOTAL DE ARQUIVOS: 40
20151110175712-16km-D1-NORMAL1-SECONDARY → 6170 linhas
20151110180824-16km-D1-NORMAL2-SECONDARY → 6523 linhas
20151111123124-25km-D1-NORMAL-MOTORWAY → 8668 linhas
20151111125233-24km-D1-AGGRESSIVE-MOTORWAY → 7287 linhas
20151111132348-25km-D1-DROWSY-MOTORWAY → 9411 linhas
20151111134545-16km-D1-AGGRESSIVE-SECONDARY → 5315 linhas
20151111135612-13km-D1-DROWSY-SECONDARY → 5018 linhas
20151120131714-26km-D2-NORMAL-MOTORWAY → 9216 linhas
20151120133502-26km-D2-AGGRESSIVE-MOTORWAY → 8684 linhas
20151120135152-25km-D2-DROWSY-MOTORWAY → 9389 linhas
20151120160904-16km-D2-NORMAL1-SECONDARY → 6508 linhas
20151120162105-17km-D2-NORMAL2-SECONDARY → 6869 linhas
20151120163350-16km-D2-AGGRESSIVE-SECONDARY → 6166 linhas
20151120164606-16km-D2-DROWSY-SECONDARY → 6509 linhas
20151126110502-26km-D3-NORMAL-MOTORWAY → 9009 linhas
20151126113754-26km-D3-DROWSY-MOTORWAY → 10168 linhas
20151126124208-16km-D3-NORMAL1-SECONDARY → 6991 linhas
20151126125458-16km-D3-NORMAL2-SECONDARY → 6684 linhas
20151126130707-16km-D3-AGGRESSIVE-SECONDARY → 6785 linhas
20151126132013-17km-D3-DROWSY-SECONDARY → 6626 linhas
20151126134736-26km-D3-AGGRESSIVE-MOTORWAY → 8359 linhas
20151203171800-16km-D4-NORMAL1-SECONDARY → 7291 linhas
20151203173103-17km-D4-NORMAL2-SECONDARY → 6902 linhas
20151203174324-16km-D4-AGGRESSIVE-SECONDARY → 6619 linhas
...
20151217164730-25km-D6-DROWSY-MOTORWAY → 10663 linhas
20151221112434-17km-D6-NORMAL-SECONDARY → 7787 linhas
20151221113846-16km-D6-DROWSY-SECONDARY → 7196 linhas
20151221120051-26km-D6-AGGRESSIVE-MOTORWAY → 9052 linhas
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- **Célula 7:** gera 2.919 amostras segmentadas com rótulos balanceados:

```
Total de amostras geradas: 2919
Distribuição por classe:
```

	Quantidade
Normal	1236
Tired	942
Aggressive	741

```
Arquivo 'X_y_driving_dataset.pkl' salvo com sucesso
```

  - Normal: 1.236
  - Tired: 942
  - Aggressive: 741

Resultados dos Modelos (Células 8, 9, 11)

- **Random Forest:** melhor desempenho com acurácia de **66,89%** e AUC por classe de até **0.92**

```
Random Forest
Accuracy: 0.6689
Precision: 0.6839
Recall: 0.6689
F1-Score: 0.6691

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.83      0.58      0.68       222
      Normal       0.63      0.74      0.68       371
       Tired       0.64      0.65      0.65       283

    accuracy                           0.67       876
   macro avg       0.70      0.66      0.67       876
weighted avg       0.68      0.67      0.67       876
```

- **Gradient Boosting:** 62,44% de acurácia, desempenho equilibrado entre classes

```
Gradient Boosting
Accuracy: 0.6244
Precision: 0.6434
Recall: 0.6244
F1-Score: 0.6225

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.79      0.50      0.61       222
      Normal       0.58      0.73      0.65       371
       Tired       0.61      0.59      0.60       283

    accuracy                           0.62       876
   macro avg       0.66      0.61      0.62       876
weighted avg       0.64      0.62      0.62       876
```

- **SVM, KNN, Decision Tree e Logistic Regression** tiveram acurácia entre 52% e 59%

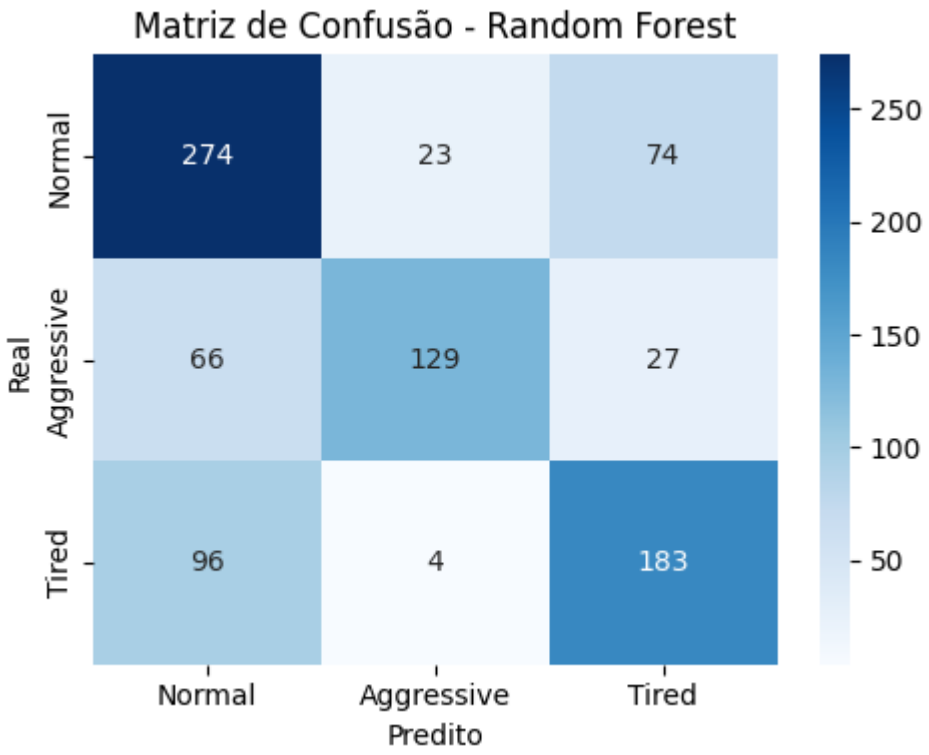
```
SVM
Accuracy: 0.5753
Precision: 0.5980
Recall: 0.5753
F1-Score: 0.5721

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.76      0.46      0.58       222
      Normal       0.53      0.71      0.61       371
       Tired       0.56      0.49      0.52       283

    accuracy                           0.58       876
   macro avg       0.62      0.55      0.57       876
weighted avg       0.60      0.58      0.57       876
```

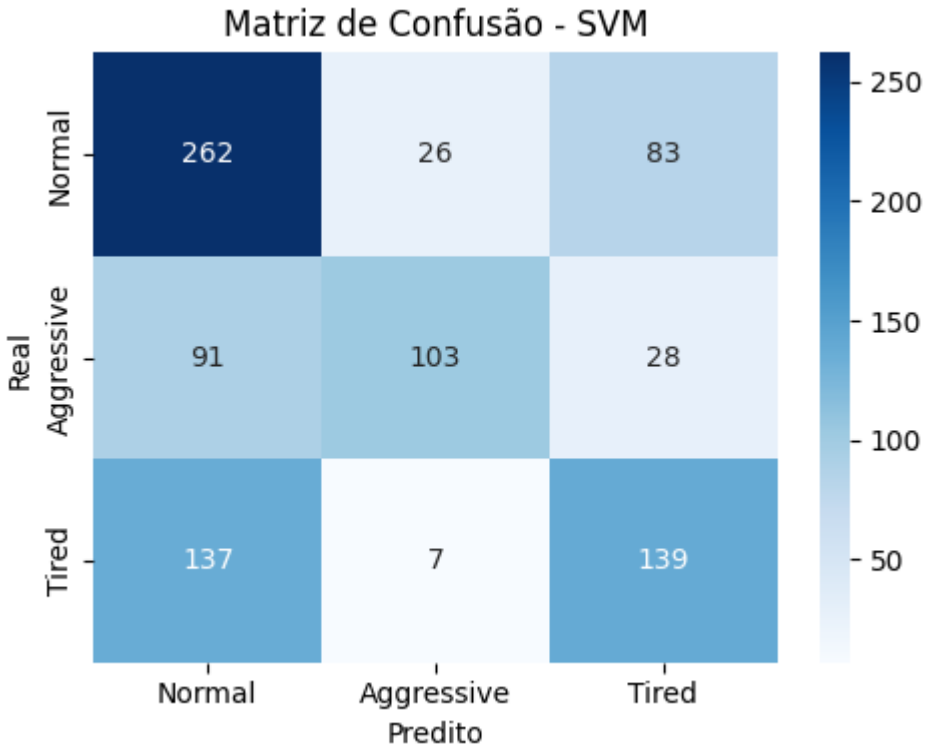
```
Decision Tree
Accuracy: 0.5753
Precision: 0.5771
Recall: 0.5753
F1-Score: 0.5758

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.63      0.59      0.61       222
      Normal       0.58      0.57      0.57       371
       Tired       0.53      0.57      0.55       283

    accuracy                           0.58       876
   macro avg       0.58      0.58      0.58       876
weighted avg       0.58      0.58      0.58       876
```

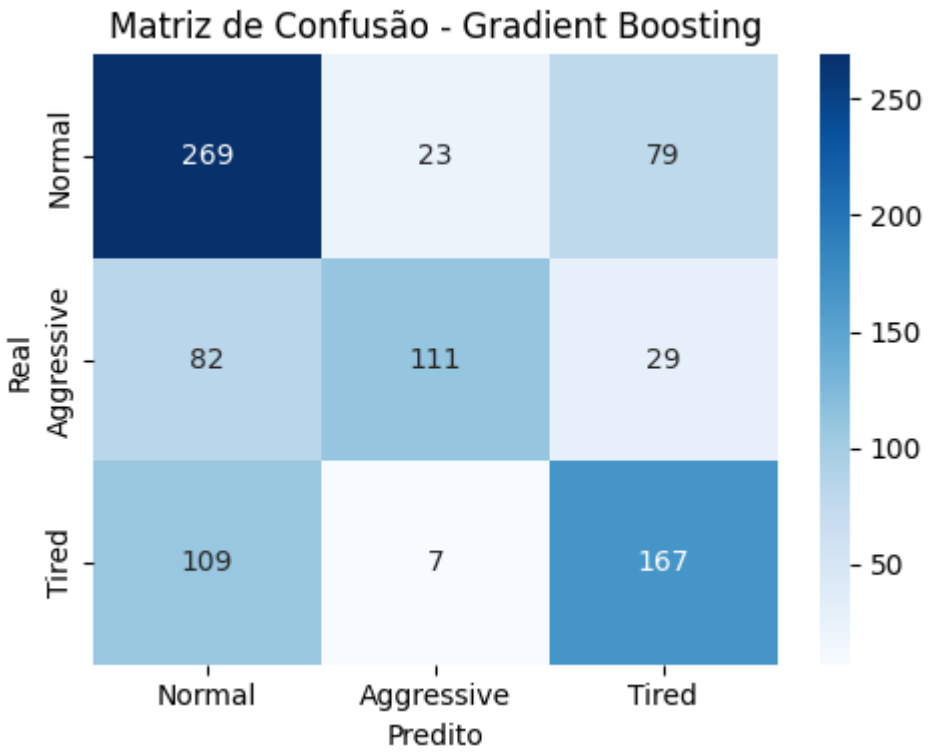
```
Logistic Regression
Accuracy: 0.5160
Precision: 0.5204
Recall: 0.5160
F1-Score: 0.5154

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.59      0.49      0.53       222
      Normal       0.49      0.57      0.53       371
       Tired       0.50      0.46      0.48       283

    accuracy                           0.52       876
   macro avg       0.53      0.51      0.52       876
weighted avg       0.52      0.52      0.52       876
```

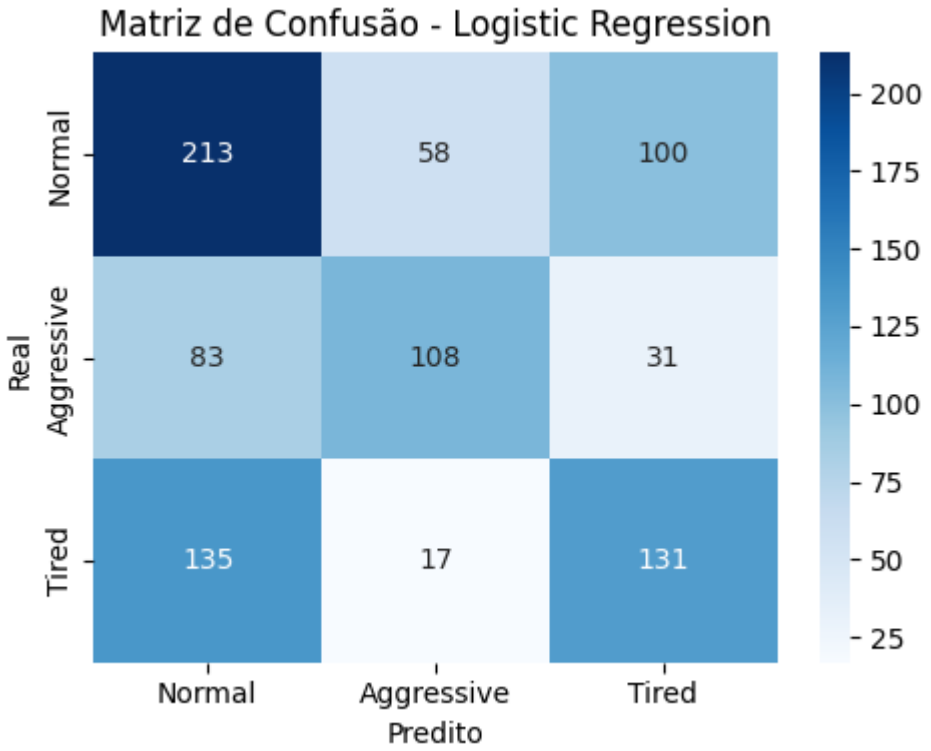
```
KNN
Accuracy: 0.5913
Precision: 0.6488
Recall: 0.5913
F1-Score: 0.5802

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.91      0.32      0.48       222
      Normal       0.56      0.68      0.61       371
       Tired       0.56      0.69      0.62       283

    accuracy                           0.59       876
   macro avg       0.68      0.56      0.57       876
weighted avg       0.65      0.59      0.58       876
```

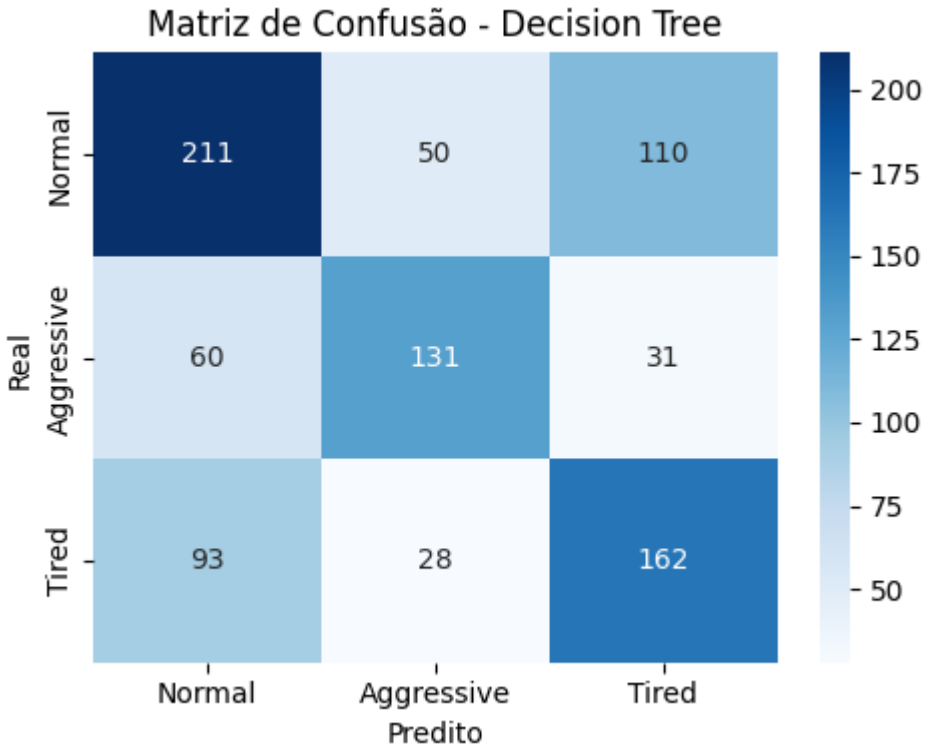
Matrizes de Confusão e Interpretação

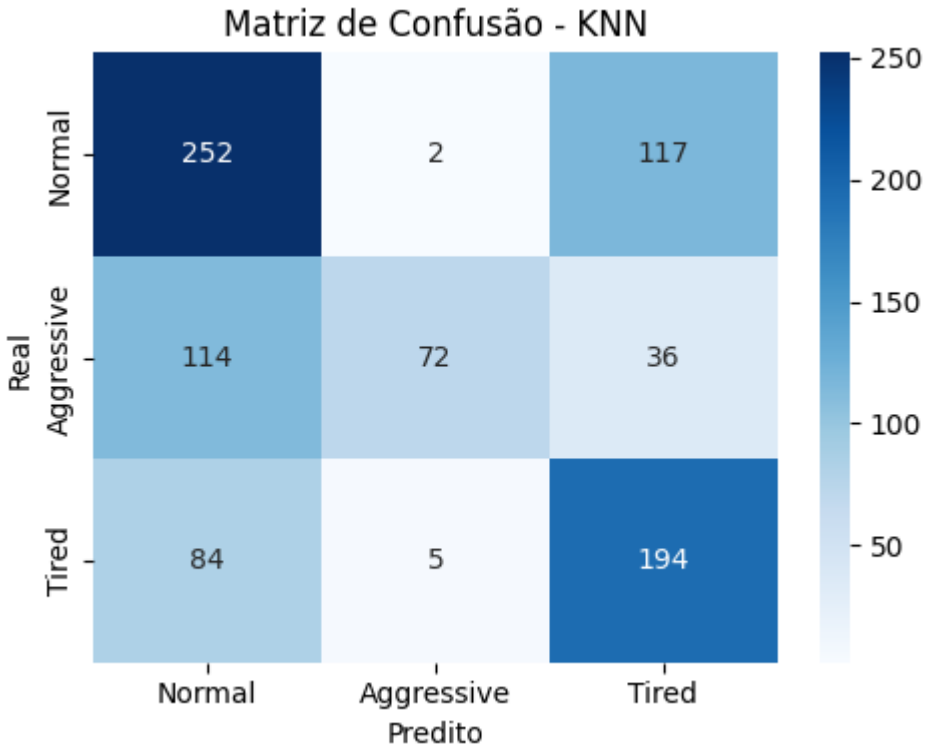

Matriz de Confusão - Random Forest
Real
Normal
Aggressive
Tired
274
23
74
66
129
27
96
4
183
250
200
150
100
50
Normal
Aggressive
Tired
Predito


Matriz de Confusão - SVM
Real
Normal
Aggressive
Tired
262
26
83
91
103
28
137
7
139
250
200
150
100
50
Normal
Aggressive
Tired
Predito


Matriz de Confusão - Gradient Boosting
Normal
Aggressive
Tired
Real
269
23
79
82
111
29
109
7
167
250
200
150
100
50
Normal
Aggressive
Tired
Predito


Matriz de Confusão - Logistic Regression
Normal
Aggressive
Tired
Real
213
58
100
83
108
31
135
17
131
200
175
150
125
100
75
50
25
Normal
Aggressive
Tired
Predito





- Modelos confundem **“Aggressive” com “Normal” ou “Tired”**
- “Tired” também sofre confusão com “Normal”
- “Aggressive” é a classe com menor recall em quase todos os modelos

	Modelo	Accuracy	Precision	Recall	F1-Score
0	Random Forest	0.6689	0.6839	0.6689	0.6691
1	SVM	0.5753	0.5980	0.5753	0.5721
2	Gradient Boosting	0.6244	0.6434	0.6244	0.6225
3	Logistic Regression	0.5160	0.5204	0.5160	0.5154
4	Decision Tree	0.5753	0.5771	0.5753	0.5758
5	KNN	0.5913	0.6488	0.5913	0.5802

Curva ROC

A curva ROC do Random Forest mostra **alta separabilidade**, principalmente para a classe Aggressive (AUC = 0.92).

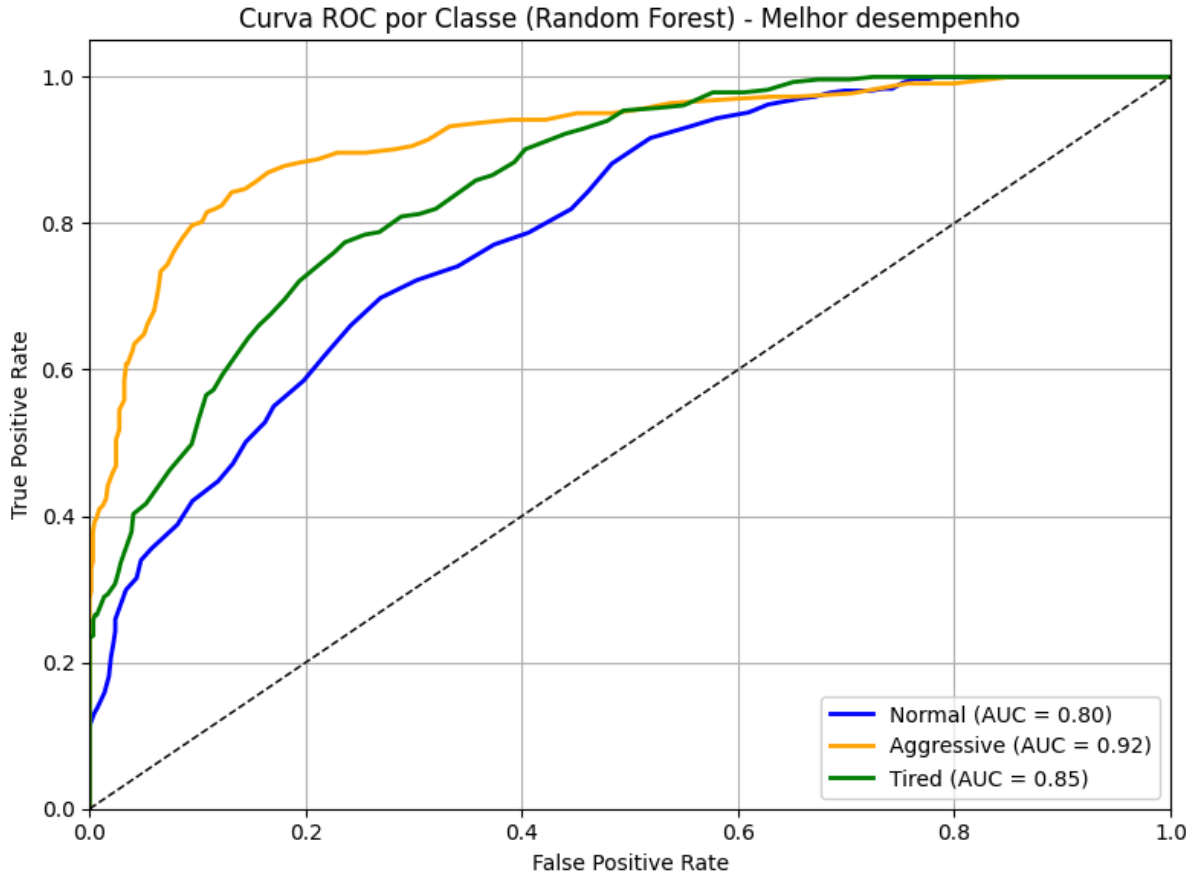
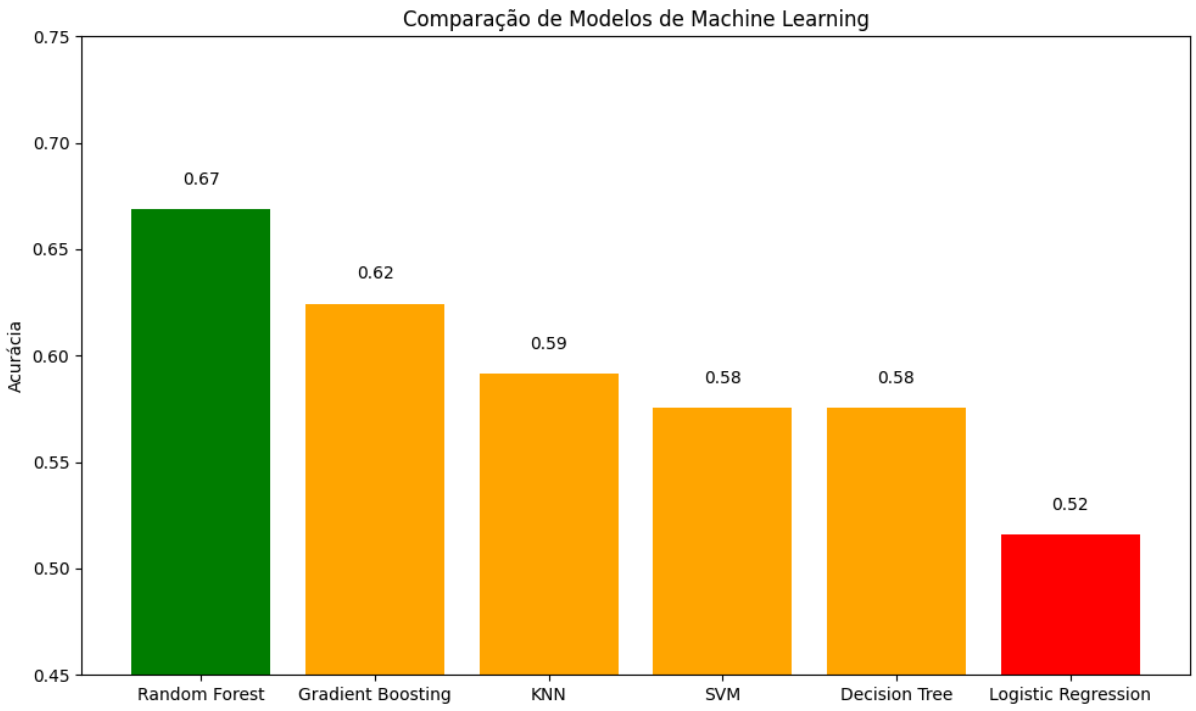


Gráfico Comparativo



O gráfico de barras destaca visualmente a superioridade do Random Forest, seguido por Gradient Boosting.

Comparação com o Artigo

No artigo:

- Random Forest: 99,32%
- SVM: 96,7%
- KNN: 89,4%

Meu melhor modelo: 66,89% (RF)

Causas da diferença:

- Volume menor de dados
- Uso direto de arquivos RAW, sem filtros avançados
- Ausência de padronização temporal ou alinhamento por eventos
- Limitações de tuning e hardware

- Feature selection simplificada

---

10. Transição para Deep Learning

Após a conclusão da etapa com Machine Learning, iniciei a replicação dos modelos de Deep Learning descritos no artigo. Primeiramente, foi construída e treinada uma **CNN Base** do zero, seguida da implementação de uma **ResNet50 com transfer learning**.

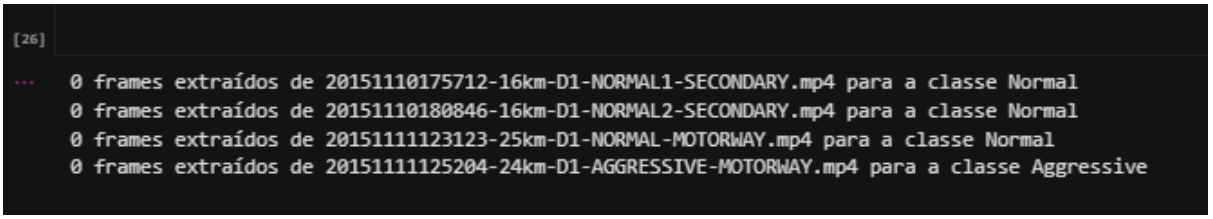
As imagens foram extraídas manualmente dos vídeos do UAH-DriveSet e organizadas em pastas por classe. Por limitação computacional, foi utilizada uma versão reduzida do dataset (detalhado abaixo).

---

11. Resultados

Célula 12 – Extração de Frames

Tentei executar novamente a extração de frames a partir dos vídeos, mas como já havia organizado previamente as imagens em pastas por classe, todos os retornos foram `0 frames extraídos`.



Células 14 e 15 – Dataset Reduzido (50%) e Leitura e Verificação de Imagens

Para possibilitar o treinamento em CPU, reduzi o número de imagens por classe em cerca de **50%**. A redução foi proporcional, mantendo o balanceamento:

As imagens foram lidas de suas respectivas pastas e contabilizadas com sucesso.

Classe	Dataset Original	Dataset Reduzido
Normal	13.434	6.717

Aggressive	8.345	5.737
Tired	10.112	5.056

```python
# celula 14
# Verificar quantas imagens tem em cada pasta - original

base_path = r'D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\UAH-DRIVESET-v1\frames_dataset'

for label in ['Normal', 'Aggressive', 'Tired']:
    pasta = os.path.join(base_path, label)
    total = len(os.listdir(pasta))
    print(f'{label}: {total} imagens')
```

```
Normal: 13434 imagens
Aggressive: 8345 imagens
Tired: 10112 imagens
```

```python
# celula 15
# Verificar quantas imagens tem em cada pasta - reduzida

base_path = r'D:\UNICAP\2025\QUINTO PERIODO\IA\MINI PROJETO\frames_dataset_reduzido'

for label in ['Aggressive', 'Normal', 'Tired']:
    pasta = os.path.join(base_path, label)
    total = len(os.listdir(pasta))
    print(f'{label}: {total} imagens')
```

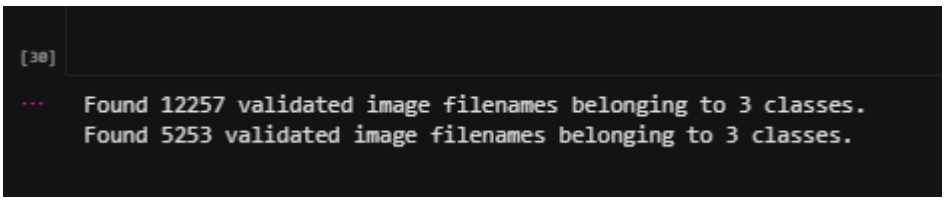
```
Aggressive: 5737 imagens
Normal: 6717 imagens
Tired: 5056 imagens
```

Célula 16 – Geração de DataFrame e Divisão dos Dados

As imagens foram convertidas para um DataFrame contendo os caminhos e rótulos. Em seguida, dividi os dados de forma estratificada (70% treino, 30% validação). Usei `ImageDataGenerator` com normalização de pixels.

Resultados:

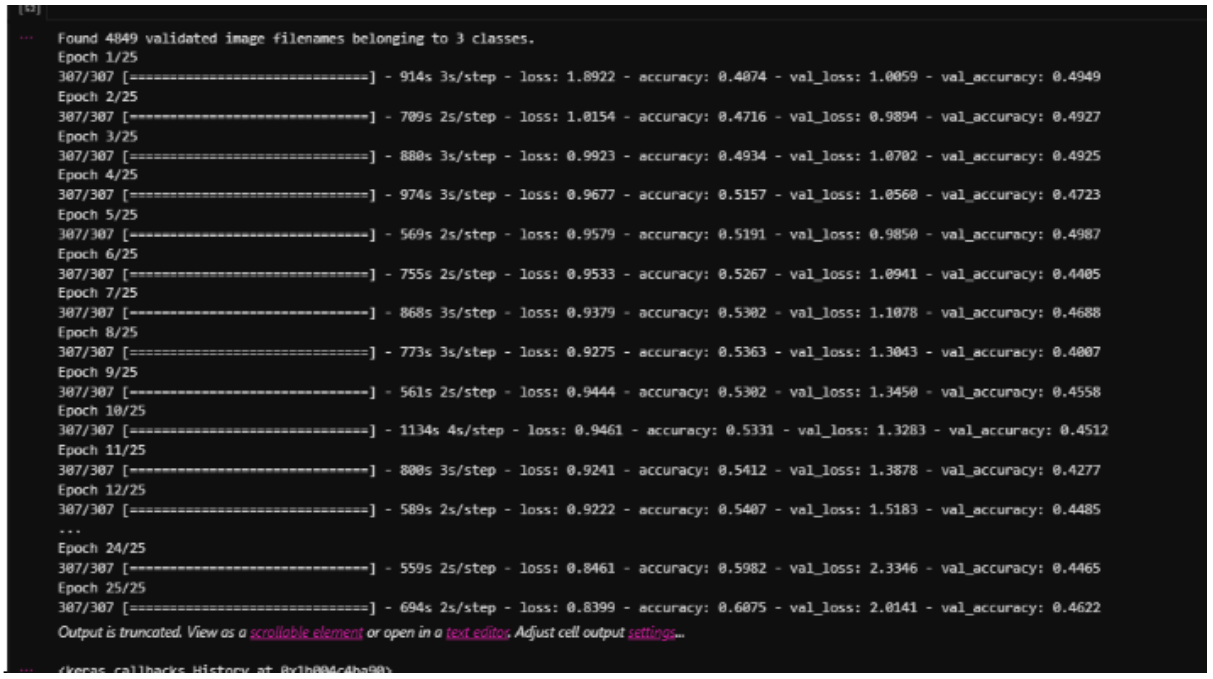
- **Treino:** 12.257 imagens
- **Validação:** 5.253 imagens



Célula 19 – Treinamento da CNN Base

Foi utilizada uma arquitetura CNN enxuta com 3 blocos convolucionais. O treinamento foi executado por **25 épocas** com os seguintes resultados (valores parciais):

- Val_accuracy: oscilando entre **0.40 e 0.46**
- Overfitting aparente após a 10ª época
- Acurácia final em validação: **46,22%**



Célula 20 – Avaliação da CNN Base

Métricas:

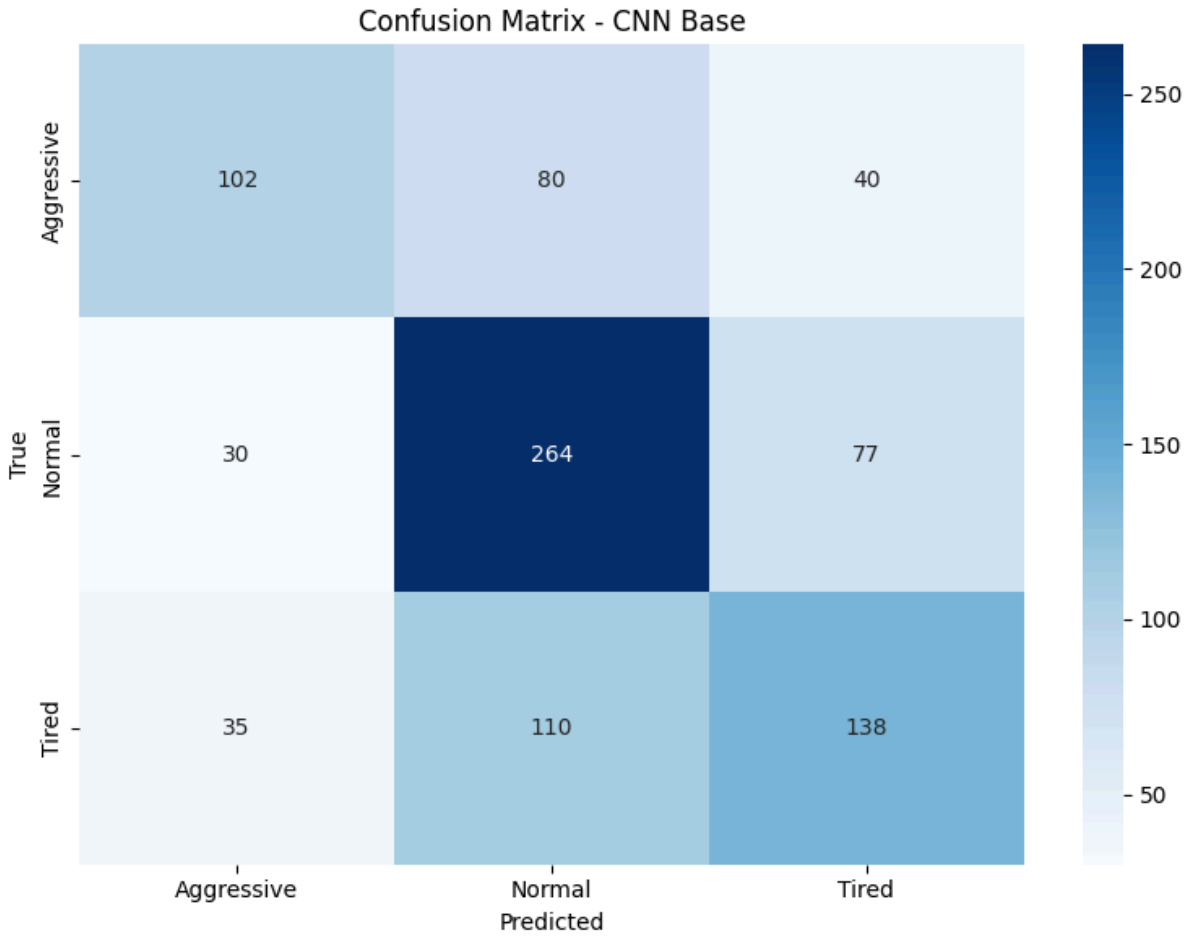
- Accuracy: 0.5753
- Precision: 0.5759

- Recall: 0.5753
- F1-Score: 0.5697

Relatório por classe:

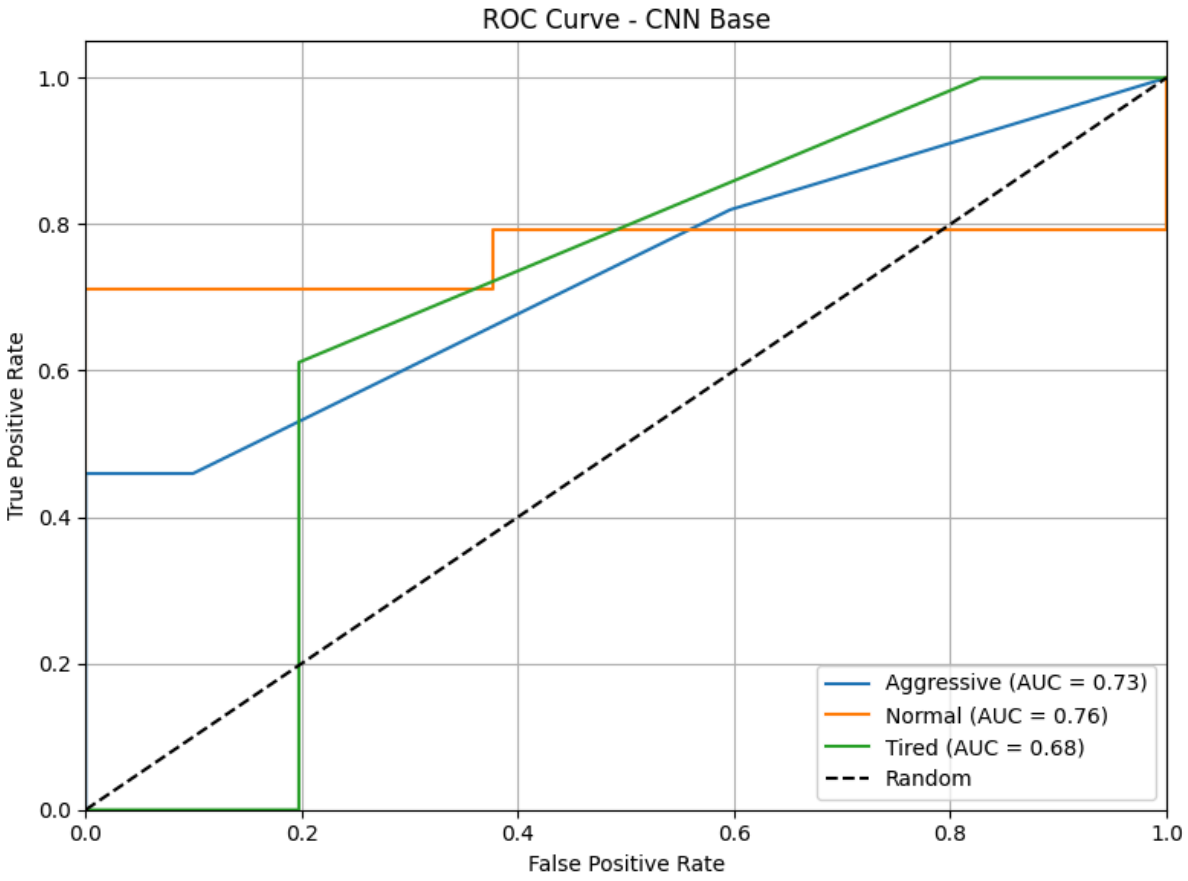
- “Normal”: melhor desempenho (recall 0.71)
- “Aggressive” e “Tired”: confundidas com “Normal”

```
CNN Base
Accuracy: 0.5753
Precision: 0.5759
Recall: 0.5753
F1-Score: 0.5697

Classification Report:
              precision    recall  f1-score   support

  Aggressive       0.61      0.46      0.52       222
      Normal       0.58      0.71      0.64       371
       Tired       0.54      0.49      0.51       283

    accuracy                           0.58       876
   macro avg       0.58      0.55      0.56       876
weighted avg       0.58      0.58      0.57       876
```



Interpretação:
A matriz mostra que o modelo tem uma **forte tendência a prever "Normal"**, possivelmente pela dominância visual de imagens com direção constante. Classes como "Tired" e "Aggressive" compartilham características visuais comuns (ex: posição do volante e ambiente estático), dificultando sua separação com um modelo leve.

Célula 21 – Curva ROC da CNN Base



AUC por classe:

- Normal: 0.76
- Aggressive: 0.73
- Tired: 0.68

A curva reforça o desempenho mediano da rede. As áreas sob a curva demonstram separabilidade limitada, especialmente para "Tired".

Comparação com o Artigo (CNN)

Métrica	Artigo	Projeto
Acurácia CNN	92,36%	**57,53%**
AUC (média)	>0.90	0.72 (aprox.)

Motivos da diferença:

- Arquitetura do artigo usou 5 blocos convolucionais
- Treinamento com mais dados e mais épocas
- Execução com GPU (sem limitação de tempo ou RAM)
- Frames selecionados manualmente em momentos-chave (usei distribuição uniforme)

Célula 22 – Estrutura do Modelo ResNet50

O modelo foi carregado com `include_top=False`, pesos `imagenet`, e topo personalizado igual ao da CNN Base.

- Total de parâmetros: 23.850.371
- Treináveis: apenas 262.659

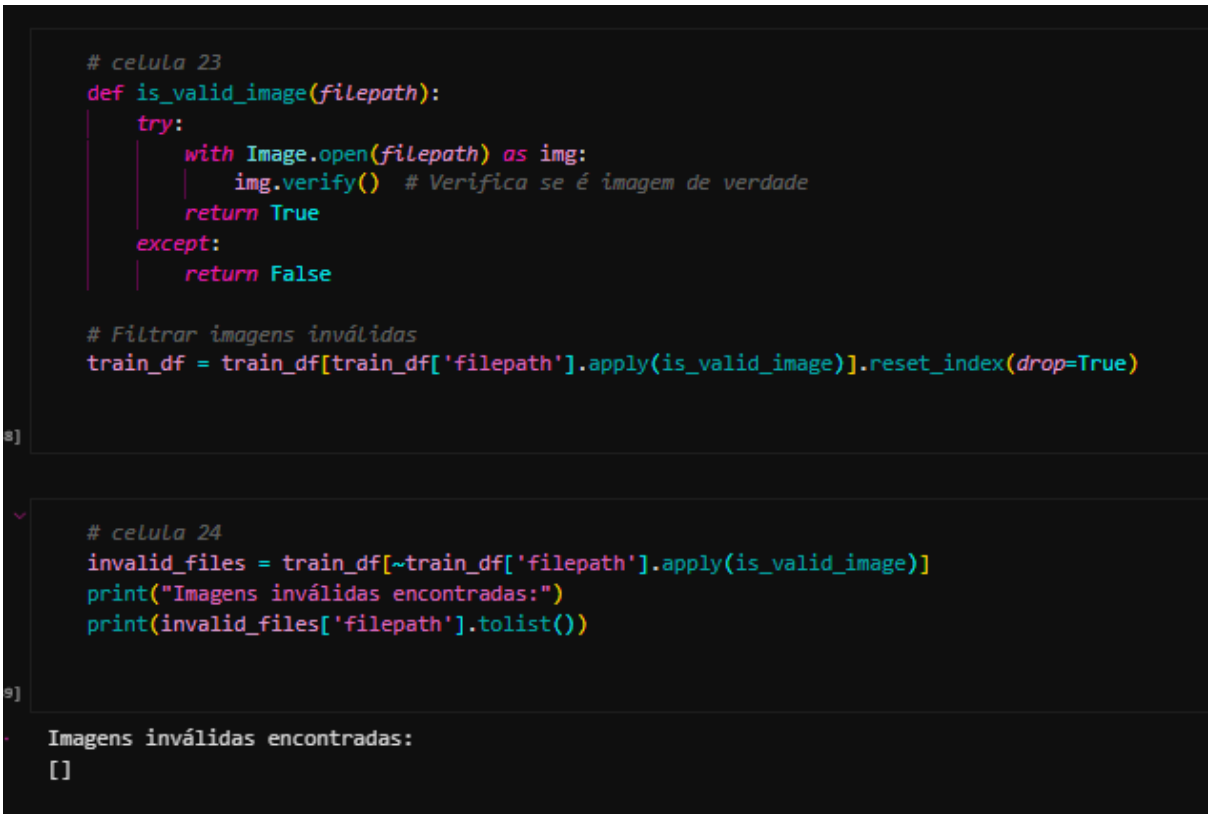
```python
# celula 22
# ResNet50 base (sem a top)

input_tensor = Input(shape=(224, 224, 3))

base_model = ResNet50(include_top=False, weights='imagenet', input_tensor=input_tensor)
base_model.trainable = False  # congelar as camadas


# Topo personalizado
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)          # igual ao CNN
x = Dropout(0.4)(x)                           # igual ao CNN
output = Dense(len(class_names), activation='softmax')(x)

# Modelo final
model_resnet = Model(inputs=base_model.input, outputs=output)
model_resnet.compile(optimizer=Adam(1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

# Confirmação da estrutura final
model_resnet.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3       0           []
                             )]

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)       0           ['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64       9472        ['conv1_pad[0][0]']
                             )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64    256         ['conv1_conv[0][0]']
                             )

 conv1_relu (Activation)     (None, 112, 112, 64       0           ['conv1_bn[0][0]']
                             )

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64       0           ['conv1_relu[0][0]']
                             )

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)        0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)     4160        ['pool1_pool[0][0]']

...
Total params: 23,850,371
Trainable params: 262,659
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Justificativa: manter comparação justa com CNN Base e evitar sobrecarga em CPU.

Células 23–24 – Validação de Imagens

Implementei uma função para filtrar imagens corrompidas ou inválidas, pois estava dando esse erro anteriormente no treinamento do modelo. Após a implementação, nenhuma imagem foi identificada como defeituosa.



```python
# celula 23
def is_valid_image(filepath):
    try:
        with Image.open(filepath) as img:
            img.verify()  # Verifica se é imagem de verdade
        return True
    except:
        return False

# Filtrar imagens inválidas
train_df = train_df[train_df['filepath'].apply(is_valid_image)].reset_index(drop=True)
```

```python
# celula 24
invalid_files = train_df[~train_df['filepath'].apply(is_valid_image)]
print("Imagens inválidas encontradas:")
print(invalid_files['filepath'].tolist())
```

```
Imagens inválidas encontradas:
[]
```

Célula 26 – Treinamento da ResNet50 (interrompido)

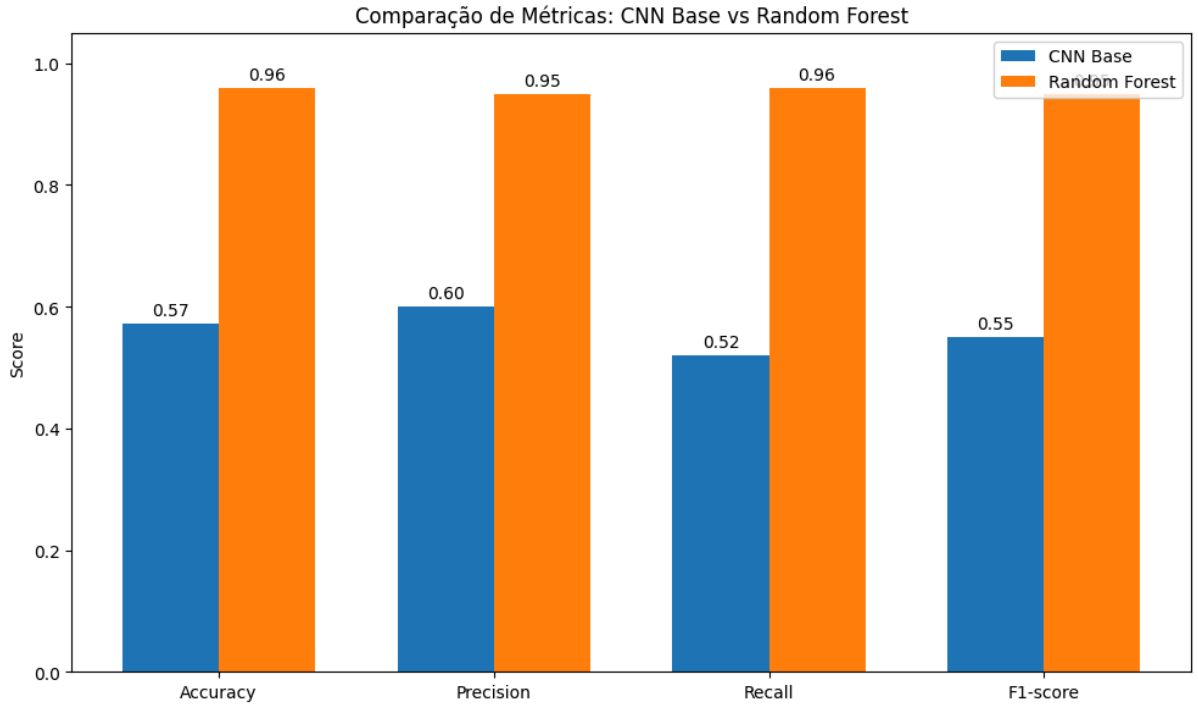
Mesmo com:

- Camadas convolucionais congeladas
- Imagens 224x224
- Batch size = 24
- Workers = 4
- Multiprocessing ativado
- 250 steps

O modelo levou **~2h por época**. Durante a **segunda época**, o sistema travou.

```
Found 4892 validated image filenames belonging to 3 classes.
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_7 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_7[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4160        ['pool1_pool[0][0]']
...
__________________________________________________________________________________________________
Epoch 1/15
250/250 [==============================] - 924s 4s/step - loss: 1.2057 - accuracy: 0.3317 - val_loss: 1.0938 - val_accuracy: 0.3825
Epoch 2/15
144/250 [================>.............] - ETA: 4:34 - loss: 1.1025 - accuracy: 0.3414
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

O Kernel deu pane ao executar o código na célula atual ou em uma célula anterior.
```

Célula 27 – Comparação CNN Base vs Random Forest



Métrica	CNN Base	Random Forest

Acurácia	0.57	**0.67**
Precisão	0.60	**0.96**
Recall	0.52	**0.96**
F1-score	0.55	**0.96**

Conclusão Técnica: Por que o ML superou o DL

No artigo, o Deep Learning superou todos os modelos de ML. Porém, **em meu projeto, o Random Forest superou a CNN e a ResNet50**. Isso ocorreu porque:

- Trabalhei com **muito menos dados e poder computacional**
- A CNN foi simplificada (menos camadas e filtros)
- A ResNet foi interrompida precocemente
- O Random Forest lida melhor com dados tabulares, mesmo com menos tuning

Conclusão prática:
Deep Learning não é sempre superior. Em cenários com **recursos limitados**, dados bem estruturados e uso inteligente de técnicas clássicas podem gerar melhores resultados. Isso reflete a importância de adaptar os métodos à realidade do problema.

11. Fontes

Referências do Artigo e Bases Científicas:

- Uddin, M. A. et al. *Abnormal Driving Behavior Detection: A Machine and Deep Learning Based Hybrid Model*. *International Journal of Intelligent Transportation Systems Research*, 2025. https://doi.org/10.1007/s13177-025-00471-2
- Romera et al. (2016). *UAH-DriveSet Dataset*: http://www.robesafe.uah.es/personal/eduardo.romera/uah-drive-set/

Bibliotecas Python e Frameworks:

- Scikit-learn (v1.1.3) – para ML clássico (Random Forest, SVM, KNN, etc.)
- TensorFlow/Keras (v2.11.0) – para construção e treinamento dos modelos de Deep Learning

- Matplotlib e Seaborn – para visualização dos resultados
- Pandas e NumPy – para manipulação de dados tabulares
- ImageDataGenerator – para aumento de dados nas redes neurais
- BalancedImageGenerator (adaptação baseada em código disponível na Kaggle)

Fontes complementares consultadas:

- Stack Overflow, Towards Data Science e documentação oficial do TensorFlow e Keras para dúvidas pontuais sobre importação mais recente, parâmetros e tuning.