relatorio_maos_de_poquer

January 8, 2026

0.1 Tópicos Especiais em Inteligência Artificial

0.2 Atividade: Classificação de mãos de poquer com Deep Neural Network (DNN)

0.2.1 Aluno: Víctor Hordones Ramos

0.2.2 Professor: Ciniro Aparecido Leite Nametala

Neste trabalho, utilizaremos o Poker Hand Data Set disponível no repositório de datasets da UCI. Este é um conjunto de dados multivariado que contém instâncias de mãos de baralho padrão, utilizado para classificar a categoria da mão de um jogador: 0 (Nothin in hand) até 9 (Royal flush). O dataset completo contém 1.025.010 amostras, mas para esta atividade utilizaremos um subconjunto de 25.010 amostras.

O modelo recebe 10 atributos de entrada (características das 5 cartas) e 1 atributo de saída (classe da mão):

1. S1: Naipe da carta #1 (1-4: Copas, Espadas, Ouros, Paus)
2. C1: Valor da carta #1 (1-13: Ás, 2, 3, …, Valete, Dama, Rei)
3. S2: Naipe da carta #2
4. C2: Valor da carta #2
5. S3: Naipe da carta #3
6. C3: Valor da carta #3
7. S4: Naipe da carta #4
8. C4: Valor da carta #4
9. S5: Naipe da carta #5
10. C5: Valor da carta #5

[SAÍDA] Classe: Classificação da mão (0 a 9, onde 0 é “Nada” e 9 é “Royal Flush”)

0.2.3 1. Importação das bibliotecas

[1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
2026-01-08 19:38:57.034599: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-01-08 19:38:57.064465: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2026-01-08 19:38:57.064488: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2026-01-08 19:38:57.065107: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2026-01-08 19:38:57.069481: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI
FMA, in other operations, rebuild TensorFlow with the appropriate compiler
flags.
```

0.2.4 2. Dataset

[2]:
```
col_names = ['s1', 'c1', 's2', 'c2', 's3', 'c3', 's4', 'c4', 's5', 'c5',␣
 ↪'class']
data_train = pd.read_csv('dataset_poquer/data_poker_treino.data', header=None,␣
 ↪names=col_names)
data_test = pd.read_csv('dataset_poquer/data_poker_teste.data', header=None,␣
 ↪names=col_names)
data_full = pd.concat([data_train, data_test], ignore_index=True)
```

[3]:
```
naipes = {1: [1, 0], 2: [1, 1], 3: [0, 0], 4: [0, 1]}
valores = {i: [int(b) for b in format(i-1, '04b')] for i in range(1, 14)}

def mapear_atributos_otimizado(data):
    lista_de_atributos = []
```

```
    for i in range(1, 6):
        s_col = data[f's{i}'].map(naipes).tolist()
        c_col = data[f'c{i}'].map(valores).tolist()
        atributos_carta = [s + c for s, c in zip(s_col, c_col)]
        lista_de_atributos.append(atributos_carta)

    X_final = [sum(cards, []) for cards in zip(*lista_de_atributos)]
    return np.array(X_final, dtype=np.int8)

X = mapear_atributos_otimizado(data_full)
y = to_categorical(data_full['class'], num_classes=10)
```

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05,␣
↪random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪05, random_state=42)
```

0.2.5 3. Rede Neural

[5]:
```
inputs = Input(shape=(30,))
x = Dense(128, activation='relu')(inputs)
x = Dropout(0.2)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.1)(x)
outputs = Dense(10, activation='softmax')(x)

model = Model(inputs, outputs)
```

```
2026-01-08 19:39:14.056715: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.101112: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.101464: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
```

```
pci#L344-L355
2026-01-08 19:39:14.102603: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.102913: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.103093: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.147876: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.148063: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.148188: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:901] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2026-01-08 19:39:14.148297: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1929] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 2798 MB memory:  -> device: 0,
name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5
```

0.2.6 3.1. Inspeção do modelo

[6]:
```python
print(model.summary())
```

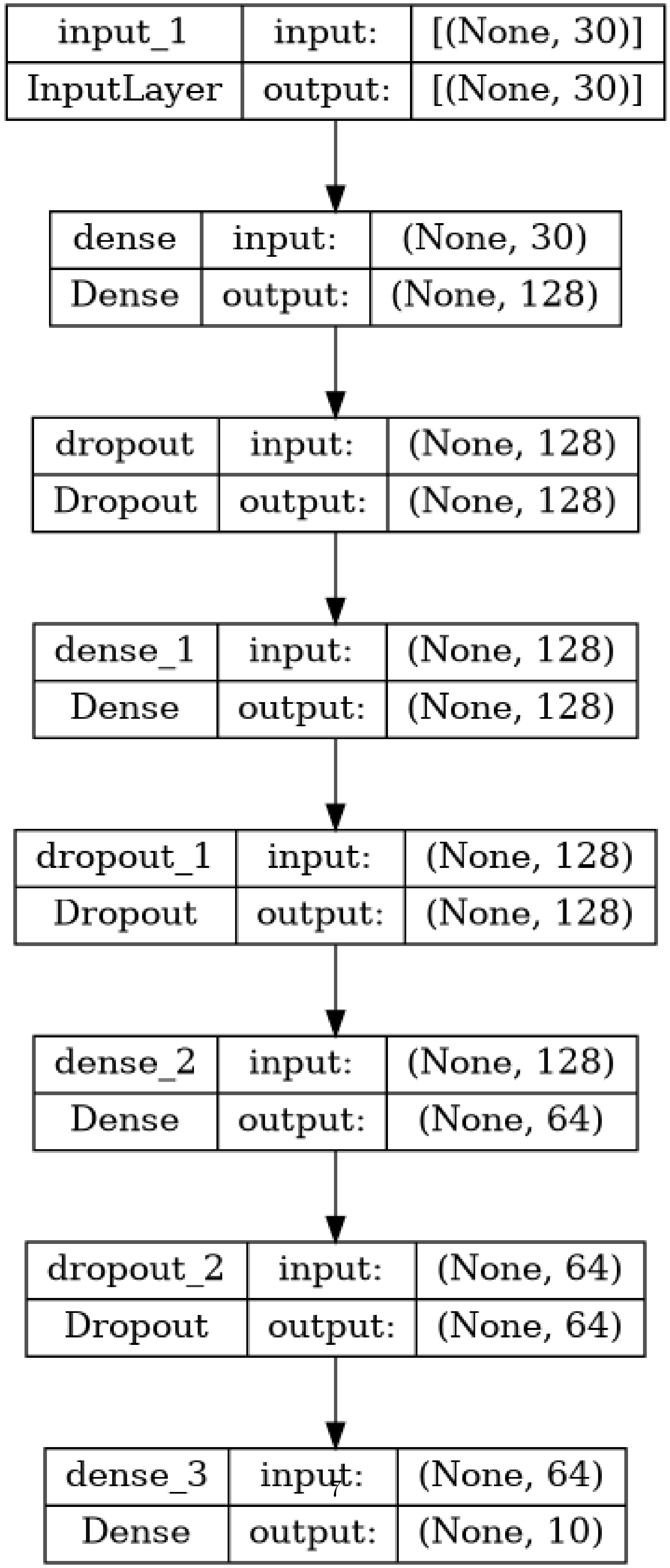
```
Model: "model"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0         
                                                                 
 dense (Dense)               (None, 128)               3968      
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 128)               16512     
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 64)                8256      
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_3 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 29386 (114.79 KB)
Trainable params: 29386 (114.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0         
                                                                 
 dense (Dense)               (None, 128)               3968      
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 128)               16512     
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_2 (Dense)             (None, 64)                8256      
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_3 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 29386 (114.79 KB)
Trainable params: 29386 (114.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

None

[7]:
```
#plotar o modelo
plot_model(model, show_shapes=True, show_layer_names=True, rankdir="TB")␣
↪#rankdir - orientacao (vertical, horizontal)
```

[7]:


input_1 | input: | [(None, 30)]
InputLayer | output: | [(None, 30)]
dense | input: | (None, 30)
Dense | output: | (None, 128)
dropout | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_2 | input: | (None, 128)
Dense | output: | (None, 64)
dropout_2 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_3 | input: | (None, 64)
Dense | output: | (None, 10)

[8]:
```
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[9]:
```
es = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=200,
    batch_size=2048,
    callbacks=[es],
    verbose=1
)
```

```
Epoch 1/200
```

```
2026-01-08 19:39:27.960170: I external/local_xla/xla/service/service.cc:168] XLA
service 0x721cc11122f0 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
2026-01-08 19:39:27.960193: I external/local_xla/xla/service/service.cc:176]
StreamExecutor device (0): NVIDIA GeForce GTX 1650, Compute Capability 7.5
2026-01-08 19:39:27.964379: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2026-01-08 19:39:27.977044: I
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:454] Loaded cuDNN
version 8907
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1767911968.029913   16542 device_compiler.h:186] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.
```

```
452/452 [==============================] - 3s 3ms/step - loss: 0.9980 -
accuracy: 0.5202 - val_loss: 0.8971 - val_accuracy: 0.5966
Epoch 2/200
452/452 [==============================] - 1s 3ms/step - loss: 0.8910 -
accuracy: 0.5968 - val_loss: 0.8464 - val_accuracy: 0.6194
Epoch 3/200
452/452 [==============================] - 1s 2ms/step - loss: 0.8419 -
accuracy: 0.6255 - val_loss: 0.7633 - val_accuracy: 0.6705
Epoch 4/200
452/452 [==============================] - 1s 2ms/step - loss: 0.7524 -
accuracy: 0.6773 - val_loss: 0.6412 - val_accuracy: 0.7327
Epoch 5/200
452/452 [==============================] - 1s 2ms/step - loss: 0.6495 -
```

```
accuracy: 0.7325 - val_loss: 0.5034 - val_accuracy: 0.7965
Epoch 6/200
452/452 [==============================] - 1s 2ms/step - loss: 0.5579 -
accuracy: 0.7781 - val_loss: 0.3990 - val_accuracy: 0.8466
Epoch 7/200
452/452 [==============================] - 1s 2ms/step - loss: 0.4939 -
accuracy: 0.8076 - val_loss: 0.3199 - val_accuracy: 0.8824
Epoch 8/200
452/452 [==============================] - 1s 3ms/step - loss: 0.4437 -
accuracy: 0.8304 - val_loss: 0.2650 - val_accuracy: 0.9013
Epoch 9/200
452/452 [==============================] - 1s 2ms/step - loss: 0.4019 -
accuracy: 0.8485 - val_loss: 0.2268 - val_accuracy: 0.9172
Epoch 10/200
452/452 [==============================] - 1s 2ms/step - loss: 0.3577 -
accuracy: 0.8686 - val_loss: 0.1562 - val_accuracy: 0.9578
Epoch 11/200
452/452 [==============================] - 1s 2ms/step - loss: 0.3111 -
accuracy: 0.8880 - val_loss: 0.1121 - val_accuracy: 0.9746
Epoch 12/200
452/452 [==============================] - 1s 2ms/step - loss: 0.2625 -
accuracy: 0.9086 - val_loss: 0.0761 - val_accuracy: 0.9863
Epoch 13/200
452/452 [==============================] - 1s 2ms/step - loss: 0.2220 -
accuracy: 0.9257 - val_loss: 0.0564 - val_accuracy: 0.9921
Epoch 14/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1900 -
accuracy: 0.9387 - val_loss: 0.0445 - val_accuracy: 0.9930
Epoch 15/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1696 -
accuracy: 0.9469 - val_loss: 0.0390 - val_accuracy: 0.9936
Epoch 16/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1549 -
accuracy: 0.9525 - val_loss: 0.0364 - val_accuracy: 0.9937
Epoch 17/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1422 -
accuracy: 0.9567 - val_loss: 0.0310 - val_accuracy: 0.9936
Epoch 18/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1303 -
accuracy: 0.9604 - val_loss: 0.0290 - val_accuracy: 0.9939
Epoch 19/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1199 -
accuracy: 0.9642 - val_loss: 0.0278 - val_accuracy: 0.9938
Epoch 20/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1119 -
accuracy: 0.9671 - val_loss: 0.0265 - val_accuracy: 0.9944
Epoch 21/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1069 -
```

```
accuracy: 0.9686 - val_loss: 0.0263 - val_accuracy: 0.9942
Epoch 22/200
452/452 [==============================] - 1s 2ms/step - loss: 0.1013 -
accuracy: 0.9706 - val_loss: 0.0253 - val_accuracy: 0.9948
Epoch 23/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0968 -
accuracy: 0.9722 - val_loss: 0.0239 - val_accuracy: 0.9952
Epoch 24/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0935 -
accuracy: 0.9734 - val_loss: 0.0229 - val_accuracy: 0.9953
Epoch 25/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0894 -
accuracy: 0.9746 - val_loss: 0.0220 - val_accuracy: 0.9956
Epoch 26/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0863 -
accuracy: 0.9757 - val_loss: 0.0221 - val_accuracy: 0.9946
Epoch 27/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0848 -
accuracy: 0.9763 - val_loss: 0.0220 - val_accuracy: 0.9954
Epoch 28/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0815 -
accuracy: 0.9775 - val_loss: 0.0203 - val_accuracy: 0.9956
Epoch 29/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0792 -
accuracy: 0.9783 - val_loss: 0.0203 - val_accuracy: 0.9962
Epoch 30/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0780 -
accuracy: 0.9787 - val_loss: 0.0195 - val_accuracy: 0.9960
Epoch 31/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0756 -
accuracy: 0.9796 - val_loss: 0.0199 - val_accuracy: 0.9961
Epoch 32/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0735 -
accuracy: 0.9803 - val_loss: 0.0201 - val_accuracy: 0.9961
Epoch 33/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0708 -
accuracy: 0.9808 - val_loss: 0.0190 - val_accuracy: 0.9962
Epoch 34/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0705 -
accuracy: 0.9812 - val_loss: 0.0184 - val_accuracy: 0.9963
Epoch 35/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0687 -
accuracy: 0.9816 - val_loss: 0.0181 - val_accuracy: 0.9966
Epoch 36/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0673 -
accuracy: 0.9822 - val_loss: 0.0174 - val_accuracy: 0.9968
Epoch 37/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0641 -
```

```
accuracy: 0.9830 - val_loss: 0.0170 - val_accuracy: 0.9970
Epoch 38/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0641 -
accuracy: 0.9831 - val_loss: 0.0176 - val_accuracy: 0.9968
Epoch 39/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0627 -
accuracy: 0.9836 - val_loss: 0.0161 - val_accuracy: 0.9972
Epoch 40/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0609 -
accuracy: 0.9838 - val_loss: 0.0170 - val_accuracy: 0.9968
Epoch 41/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0607 -
accuracy: 0.9840 - val_loss: 0.0160 - val_accuracy: 0.9967
Epoch 42/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0586 -
accuracy: 0.9847 - val_loss: 0.0155 - val_accuracy: 0.9971
Epoch 43/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0576 -
accuracy: 0.9851 - val_loss: 0.0159 - val_accuracy: 0.9969
Epoch 44/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0563 -
accuracy: 0.9854 - val_loss: 0.0155 - val_accuracy: 0.9969
Epoch 45/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0558 -
accuracy: 0.9854 - val_loss: 0.0148 - val_accuracy: 0.9971
Epoch 46/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0544 -
accuracy: 0.9859 - val_loss: 0.0143 - val_accuracy: 0.9976
Epoch 47/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0534 -
accuracy: 0.9862 - val_loss: 0.0145 - val_accuracy: 0.9979
Epoch 48/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0534 -
accuracy: 0.9865 - val_loss: 0.0145 - val_accuracy: 0.9980
Epoch 49/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0520 -
accuracy: 0.9870 - val_loss: 0.0143 - val_accuracy: 0.9975
Epoch 50/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0513 -
accuracy: 0.9873 - val_loss: 0.0142 - val_accuracy: 0.9984
Epoch 51/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0502 -
accuracy: 0.9875 - val_loss: 0.0139 - val_accuracy: 0.9985
Epoch 52/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0501 -
accuracy: 0.9876 - val_loss: 0.0139 - val_accuracy: 0.9984
Epoch 53/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0496 -
```

```
accuracy: 0.9879 - val_loss: 0.0136 - val_accuracy: 0.9985
Epoch 54/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0490 -
accuracy: 0.9882 - val_loss: 0.0140 - val_accuracy: 0.9976
Epoch 55/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0499 -
accuracy: 0.9879 - val_loss: 0.0143 - val_accuracy: 0.9983
Epoch 56/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0486 -
accuracy: 0.9882 - val_loss: 0.0137 - val_accuracy: 0.9983
Epoch 57/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0483 -
accuracy: 0.9885 - val_loss: 0.0137 - val_accuracy: 0.9984
Epoch 58/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0472 -
accuracy: 0.9886 - val_loss: 0.0140 - val_accuracy: 0.9983
Epoch 59/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0470 -
accuracy: 0.9888 - val_loss: 0.0132 - val_accuracy: 0.9984
Epoch 60/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0469 -
accuracy: 0.9888 - val_loss: 0.0134 - val_accuracy: 0.9978
Epoch 61/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0472 -
accuracy: 0.9888 - val_loss: 0.0146 - val_accuracy: 0.9980
Epoch 62/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0457 -
accuracy: 0.9891 - val_loss: 0.0143 - val_accuracy: 0.9975
Epoch 63/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0460 -
accuracy: 0.9892 - val_loss: 0.0132 - val_accuracy: 0.9985
Epoch 64/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0453 -
accuracy: 0.9893 - val_loss: 0.0133 - val_accuracy: 0.9984
Epoch 65/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0453 -
accuracy: 0.9894 - val_loss: 0.0134 - val_accuracy: 0.9983
Epoch 66/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0450 -
accuracy: 0.9895 - val_loss: 0.0142 - val_accuracy: 0.9972
Epoch 67/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0436 -
accuracy: 0.9897 - val_loss: 0.0142 - val_accuracy: 0.9982
Epoch 68/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0448 -
accuracy: 0.9895 - val_loss: 0.0136 - val_accuracy: 0.9985
Epoch 69/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0444 -
```

```
accuracy: 0.9897 - val_loss: 0.0134 - val_accuracy: 0.9979
Epoch 70/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0438 -
accuracy: 0.9898 - val_loss: 0.0129 - val_accuracy: 0.9986
Epoch 71/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0435 -
accuracy: 0.9899 - val_loss: 0.0129 - val_accuracy: 0.9979
Epoch 72/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0438 -
accuracy: 0.9899 - val_loss: 0.0139 - val_accuracy: 0.9979
Epoch 73/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0429 -
accuracy: 0.9900 - val_loss: 0.0138 - val_accuracy: 0.9976
Epoch 74/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0423 -
accuracy: 0.9903 - val_loss: 0.0136 - val_accuracy: 0.9980
Epoch 75/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0433 -
accuracy: 0.9899 - val_loss: 0.0151 - val_accuracy: 0.9970
Epoch 76/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0427 -
accuracy: 0.9900 - val_loss: 0.0153 - val_accuracy: 0.9968
Epoch 77/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0423 -
accuracy: 0.9901 - val_loss: 0.0131 - val_accuracy: 0.9980
Epoch 78/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0422 -
accuracy: 0.9903 - val_loss: 0.0136 - val_accuracy: 0.9974
Epoch 79/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0411 -
accuracy: 0.9906 - val_loss: 0.0145 - val_accuracy: 0.9968
Epoch 80/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0414 -
accuracy: 0.9904 - val_loss: 0.0138 - val_accuracy: 0.9971
Epoch 81/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0414 -
accuracy: 0.9905 - val_loss: 0.0135 - val_accuracy: 0.9979
Epoch 82/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0415 -
accuracy: 0.9904 - val_loss: 0.0156 - val_accuracy: 0.9967
Epoch 83/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0414 -
accuracy: 0.9905 - val_loss: 0.0148 - val_accuracy: 0.9970
Epoch 84/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0404 -
accuracy: 0.9906 - val_loss: 0.0136 - val_accuracy: 0.9979
Epoch 85/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0414 -
```

```
accuracy: 0.9905 - val_loss: 0.0135 - val_accuracy: 0.9972
Epoch 86/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0409 -
accuracy: 0.9905 - val_loss: 0.0145 - val_accuracy: 0.9971
Epoch 87/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0404 -
accuracy: 0.9906 - val_loss: 0.0142 - val_accuracy: 0.9972
Epoch 88/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0401 -
accuracy: 0.9907 - val_loss: 0.0144 - val_accuracy: 0.9968
Epoch 89/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0399 -
accuracy: 0.9909 - val_loss: 0.0138 - val_accuracy: 0.9970
Epoch 90/200
452/452 [==============================] - 1s 2ms/step - loss: 0.0416 -
accuracy: 0.9905 - val_loss: 0.0132 - val_accuracy: 0.9974
```

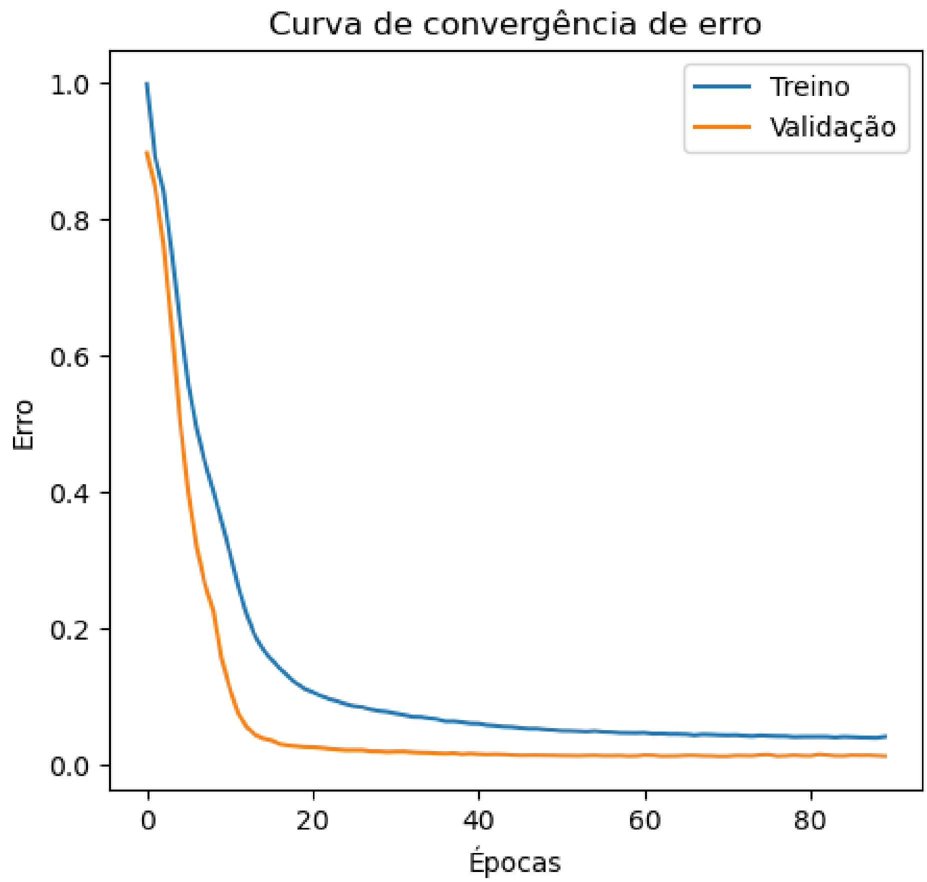
0.2.7 4. Resultados

4.1. Curva de Convergência do erro

[12]:
```python
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Acurácia no dataset de teste: {acc*100:.2f}%")

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Treino')
plt.plot(history.history['val_loss'], label='Validação')
plt.title('Curva de convergência de erro')
plt.xlabel('Épocas')
plt.ylabel('Erro')
plt.legend()
plt.show()
```

```
Acurácia no dataset de teste: 99.88%
```



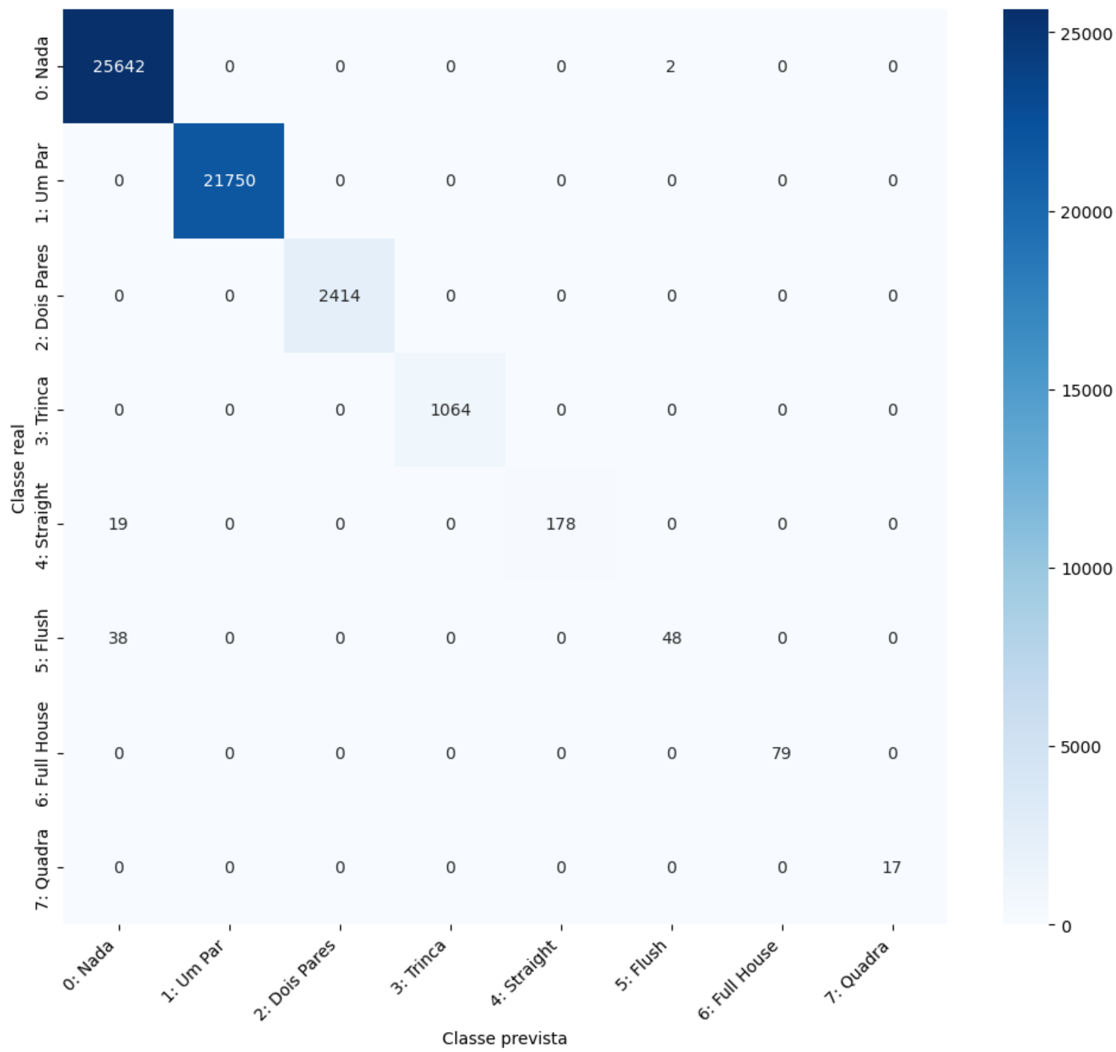
4.2. Matriz de Confusão

[11]:
```
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)
cm = confusion_matrix(y_true, y_pred)

class_names = [
    "0: Nada", "1: Um Par", "2: Dois Pares", "3: Trinca",
    "4: Straight", "5: Flush", "6: Full House", "7: Quadra"
]

plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.ylabel('Classe real')
plt.xlabel('Classe prevista')
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
1602/1602 [==============================] - 1s 755us/step
```



0.2.8 5. Perguntas e Respostas

Calcule a taxa de acerto no conjunto de teste. O que ela mostra? O modelo esta conseguindo realizar a classificação corretamente?

- Taxa de acerto: 99,88%

Essa taxa de acerto mostra que o modelo está conseguindo classificar quase todas as mãos de poquer corretamente, errando somente 0,12%.

Pela matriz de confusão:

- Nada (0): conseguindo classificar quase todas as mãos de poquer corretamente (25642 mãos), errando em apenas 2 mãos, classificou como Flush (5).
- Um par (1): conseguindo classificar todas as mãos de poquer corretamente.

- Dois pares (2): conseguindo classificar todas as mãos de poquer corretamente.
- Trinca (3): conseguindo classificar todas as mãos de poquer corretamente.
- Straight (4): conseguindo classificar quase todas as mãos de poquer corretamente (178 mãos), errando em apenas 19 mãos, * classificou como Nada (0).
- Flush (5): conseguindo classificar praticamente 50% das mãos de Flush, acertando em 48 mãos de Flush e errando em 38 mãos, classificando como Nada (0). O erro da classificação está praticamente todo concentrado aqui
- Full House (6): conseguindo classificar todas as mãos de poquer corretamente.
- Quadra (7): conseguindo classificar todas as mãos de poquer corretamente.