Projeto - Aplicação de Redes Neurais

Objetivo: O aluno deverá ser capaz de realizar uma análise completa dos dados e projetar uma rede neural para resolver problema. Avaliar os resultados obtidos através das métricas de classificação

Problema: De posse de dados que correspondem a sinais de transitórios de eletrodomésticos (sinais obtidos em uma janela de 2s ao se ligar equipamento) e que foram rotulados em 7 diferentes classes, o aluno deverá realizar os seguintes passos:

1. Carregar os dados e realizar a limpeza dos dados (se necessário)
2. Visualizar os dados para compreensão (dica: plotar 1 exemplo de cada Classe). Como na Figura 1 , abaixo, que representa um eletrodoméstico da Classe 1.
3. Como é um problema muticlasse, o aluno deverá transformar os labels para uma representação correta.
4. Preparar os dados para se apresentados à ML
5. Construir a rede neural com seus respectivos parâmetros (taxa de aprendizado, número de camadas intermediárias, número de neurônios, batch_size etc). O aluno deve propor uma estratégia para determinar esses parâmetros.
6. Testar e validar os resultados
7. Avaliar o usa de PCA (Análise de Componentes Principais) para visualização dos dados e também como speed-up da ML (para fins de classificação).
8. Conclusão

obs.: Esses dados estão desatualizados, não necessariamente representam os eletrodomésticos atuais. Mas, esses estudos ainda são atuais e continuam usando recursos de IA em suas soluções.

In [71]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import sklearn
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
```

1. Carregar os dados e realizar a limpeza dos dados (se necessário)

In [72]:
```
file = 'db.csv'
df = pd.read_csv(file)
df
```

Out[72]:

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	...	t191	t192	t193
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	1.00	-1.00	1.00
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	-1.00	1.00	0.00
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	-0.55	0.45	-0.55
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	-0.70	0.30	-0.70
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	-0.15	0.85	-1.15
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
95	14.70	4.70	2.70	-0.30	0.70	0.70	2.70	1.70	1.70	0.70	...	-0.30	-0.30	-0.30
96	11.55	5.55	2.55	1.55	0.55	1.55	0.55	0.55	-0.45	0.55	...	0.55	-0.45	0.55
97	14.35	4.35	1.35	2.35	-0.65	0.35	0.35	1.35	0.35	0.35	...	-0.65	0.35	-0.65
98	22.70	-5.30	-0.30	-5.30	-1.30	-3.30	-1.30	-3.30	-2.30	-3.30	...	-2.30	1.70	-1.30
99	-5.30	-6.30	-6.30	-6.30	-5.30	-3.30	-4.30	-6.30	-6.30	-4.30	...	-0.30	0.70	-1.30

100 rows × 201 columns

In [73]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Columns: 201 entries, t0 to Classes
dtypes: float64(200), int64(1)
memory usage: 157.2 KB
```

In [74]:
```
df.describe()
```

Out[74]:

	t0	t1	t2	t3	t4	t5	t6
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	11.114000	7.974000	8.154000	7.334000	7.374000	6.834000	7.134000
std	10.187807	9.012776	8.899937	8.958318	8.458134	8.301816	8.230796
min	-10.150000	-6.300000	-6.300000	-6.300000	-5.300000	-3.500000	-4.300000
25%	0.650000	0.450000	0.700000	0.450000	0.650000	0.450000	0.600000
50%	11.925000	5.575000	4.950000	4.300000	3.600000	3.050000	3.300000
75%	21.250000	14.900000	17.450000	16.950000	14.900000	13.000000	14.000000
max	25.850000	25.450000	26.200000	25.200000	25.200000	25.000000	24.200000

8 rows × 201 columns

In [75]:
```
df.describe().loc['count'].unique() # Com isso é possivel verificar que não
```

Out[75]:
```
array([100.])
```

In [76]:
```
y = df.iloc[:,[-1]]
y
```

Out[76]:

	Classes
0	1
1	1
2	1
3	1
4	2
...	...
95	5
96	6
97	6
98	7
99	7

100 rows × 1 columns

In [77]:
```
X = df.iloc[:, 0:-1]
X
```

Out[77]:

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	...	t190	t191	t192
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	-1.00	1.00	-1.00
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	0.00	-1.00	1.00
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	0.45	-0.55	0.45
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	0.30	-0.70	0.30
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	0.85	-0.15	0.85
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
95	14.70	4.70	2.70	-0.30	0.70	0.70	2.70	1.70	1.70	0.70	...	0.70	-0.30	-0.30
96	11.55	5.55	2.55	1.55	0.55	1.55	0.55	0.55	-0.45	0.55	...	-0.45	0.55	-0.45
97	14.35	4.35	1.35	2.35	-0.65	0.35	0.35	1.35	0.35	0.35	...	0.35	-0.65	0.35
98	22.70	-5.30	-0.30	-5.30	-1.30	-3.30	-1.30	-3.30	-2.30	-3.30	...	1.70	-2.30	1.70
99	-5.30	-6.30	-6.30	-6.30	-5.30	-3.30	-4.30	-6.30	-6.30	-4.30	...	0.70	-0.30	0.70

100 rows × 200 columns

In [78]:
```
X.columns = list(range(len(X.columns))) # alterando a nomeclatura das colun
X.head()
```

Out[78]:

	0	1	2	3	4	5	6	7	8	9	...	190	191	192
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	-1.00	1.00	-1.00
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	0.00	-1.00	1.00
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	0.45	-0.55	0.45
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	0.30	-0.70	0.30
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	0.85	-0.15	0.85

5 rows × 200 columns

In [79]:
```
df_copy = X.copy()
df_copy
```

Out[79]:

	0	1	2	3	4	5	6	7	8	9	...	190	191	192
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	-1.00	1.00	-1.00
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	0.00	-1.00	1.00
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	0.45	-0.55	0.45
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	0.30	-0.70	0.30
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	0.85	-0.15	0.85
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
95	14.70	4.70	2.70	-0.30	0.70	0.70	2.70	1.70	1.70	0.70	...	0.70	-0.30	-0.30
96	11.55	5.55	2.55	1.55	0.55	1.55	0.55	0.55	-0.45	0.55	...	-0.45	0.55	-0.45
97	14.35	4.35	1.35	2.35	-0.65	0.35	0.35	1.35	0.35	0.35	...	0.35	-0.65	0.35
98	22.70	-5.30	-0.30	-5.30	-1.30	-3.30	-1.30	-3.30	-2.30	-3.30	...	1.70	-2.30	1.70
99	-5.30	-6.30	-6.30	-6.30	-5.30	-3.30	-4.30	-6.30	-6.30	-4.30	...	0.70	-0.30	0.70

100 rows × 200 columns

In [80]:
```
df_copy['Classes'] = y
```

2. Visualizar os dados para compreensão (dica: plotar 1 exemplo de cada Classe). Como na Figura 1 , abaixo, que representa um eletrodoméstico da Classe 1.

In [81]:
```
df_copy.head()
```

Out[81]:

	0	1	2	3	4	5	6	7	8	9	...	191	192	193
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	1.00	-1.00	1.00
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	-1.00	1.00	0.00
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	-0.55	0.45	-0.55
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	-0.70	0.30	-0.70
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	-0.15	0.85	-1.15

5 rows × 201 columns

In [82]:
```
df_copy.loc[:,'Classes'].unique() # existem 7 classes diferentes
```

Out[82]:
```
array([1, 2, 3, 4, 5, 6, 7], dtype=int64)
```

In [83]:
```
df_classes = df_copy.groupby('Classes').mean()
df_classes
```

Out[83]:

Classes	0	1	2	3	4	5	6	
1	13.306250	12.681250	15.931250	16.806250	17.931250	17.181250	17.681250	19.306
2	2.227273	1.045455	1.409091	0.318182	0.954545	0.409091	0.863636	0.409
3	12.740000	12.006667	12.273333	11.073333	10.406667	9.606667	9.740000	8.873
4	8.550000	2.800000	4.300000	2.550000	3.800000	3.050000	4.550000	3.300
5	19.518750	12.018750	8.268750	7.518750	6.268750	6.018750	5.768750	5.768
6	12.950000	4.950000	1.950000	1.950000	-0.050000	0.950000	0.450000	0.950
7	8.700000	-5.800000	-3.300000	-5.800000	-3.300000	-3.300000	-2.800000	-4.800

7 rows × 200 columns

In [84]:
```
# Supondo que 'df_classes' é o seu DataFrame

# Criar uma figura com subplots
fig = make_subplots(rows=7, cols=1, subplot_titles=[f'Classe {classe}' for

# Adicionar um gráfico de linha para cada classe
for i, classe in enumerate(df_classes.index):
    fig.add_trace(go.Scatter(x=df_classes.columns, y=df_classes.loc[classe]

fig.update_layout(height=4000, width=800, title_text="Médias por Classe ao
fig.show()
```

In [85]:
```
df = df_copy
df.head()
```

Out[85]:

	0	1	2	3	4	5	6	7	8	9	...	191	192	193
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	1.00	-1.00	1.00
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	-1.00	1.00	0.00
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	-0.55	0.45	-0.55
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	-0.70	0.30	-0.70
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	-0.15	0.85	-1.15

5 rows × 201 columns

3. Como é um problema muticlasse, o aluno deverá transformar os labels para uma representação correta.

In [86]:
```
# Supondo que 'labels' é um array com os rótulos das classes
labels = df['Classes'].values.reshape(-1, 1)  # Redimensionar para um array

# Criar o codificador e transformar os rótulos
encoder = OneHotEncoder(sparse_output=False)  # sparse=False para retornar
labels_one_hot = encoder.fit_transform(labels)
```

In [87]:
```
len(labels_one_hot)
```

Out[87]:
```
100
```

In [88]:
```
labels_one_hot
```

Out[88]:
```
array([[1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
```

```
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 1.]])
```

In [89]:
```
# Separar as features e os rótulos
X = df.drop('Classes', axis=1).values  # Converter as features para um arra
y = labels_one_hot  # Já é um array numpy de rótulos one-hot
```

In [90]:
```
X[:5]
```

Out[90]:
```
array([[24.  , 24.  , 23.  , 25.  , 24.  , 25.  , 24.  , 24.  , 22.  ,
        25.  , 24.  , 25.  , 23.  , 23.  , 23.  , 24.  , 24.  , 24.  ,
        23.  , 22.  , 20.  , 22.  , 20.  , 19.  , 15.  , 12.  ,  5.  ,
         2.  ,  0.  ,  0.  ,  0.  ,  1.  , -2.  ,  0.  , -2.  ,  1.  ,
         0.  ,  1.  , -2.  ,  0.  , -1.  ,  1.  , -1.  ,  0.  , -2.  ,
         0.  , -1.  ,  1.  ,  0.  ,  1.  , -1.  ,  0.  , -1.  ,  1.  ,
        -1.  ,  0.  , -1.  ,  1.  , -1.  ,  1.  ,  0.  ,  1.  , -1.  ,
         0.  ,  0.  ,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  1.  ,
         0.  ,  0.  , -1.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,  0.  ,
         0.  ,  0.  ,  1.  ,  0.  ,  0.  , -1.  ,  0.  ,  0.  ,  1.  ,
         0.  ,  1.  , -1.  ,  1.  ,  0.  ,  1.  ,  0.  ,  1.  , -1.  ,
         0.  ,  0.  ,  1.  ,  0.  ,  1.  , -1.  ,  0.  , -1.  ,  1.  ,
        -1.  ,  1.  , -1.  ,  1.  , -1.  ,  1.  , -1.  ,  1.  , -2.  ,
         1.  , -1.  ,  1.  , -1.  ,  1.  , -1.  ,  1.  ,  0.  ,  1.  ,
         0.  ,  0.  , -1.  ,  1.  ,  0.  ,  1.  ,  0.  ,  0.  , -1.  ,
         1.  ,  0.  ,  1.  ,  0.  ,  0.  , -1.  ,  0.  ,  0.  ,  1.  ,
         0.  ,  0.  , -1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  , -1.  ,
         0.  ,  0.  ,  1.  ,  0.  ,  1.  , -1.  ,  0.  ,  0.  ,  1.  ,
         0.  ,  0.  , -1.  ,  0.  ,  0.  ,  1.  , -1.  ,  1.  , -1.  ,
```

In [91]:
```
y[:5]
```

Out[91]:
```
array([[1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.]])
```

4. Preparar os dados para se apresentados à ML

In [92]:
```
# Agora vamos separar os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [93]:
```
# Normalizando os dados
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

5. Construir a rede neural com seus respectivos parâmetros (taxa de aprendizado, número de camadas intermediárias, número de neurônios, batch_size etc). O aluno deve propor uma estratégia para determinar esses parâmetros.
6. Testar e validar os resultados

In [94]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu'),  # Primeira camada oculta com 128 neurôni
    # Dense(128, activation='relu'),  # Segunda camada oculta com 64 neurôn
    # Dense(64, activation='relu'),
    Dense(7, activation='softmax')  # Camada de saída com 7 neurônios (um p
])
```

In [95]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [96]:
```
history = model.fit(X_train, y_train,
                    epochs=70,
                    batch_size=1,
                    validation_split=0.1)  # Use parte dos dados de treino
```

```
Epoch 1/70
34/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0943 - loss: 3.074
0     72/72 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2161 - loss:
2.5802 - val_accuracy: 0.5000 - val_loss: 1.9007
Epoch 2/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8260 - loss: 0.618
8 - val_accuracy: 0.6250 - val_loss: 1.2131
Epoch 3/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9741 - loss: 0.336
8 - val_accuracy: 0.6250 - val_loss: 1.3203
Epoch 4/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9898 - loss: 0.243
1 - val_accuracy: 0.7500 - val_loss: 1.3188
Epoch 5/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9618 - loss: 0.153
9 - val_accuracy: 0.7500 - val_loss: 1.2617
Epoch 6/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.092
8 - val_accuracy: 0.7500 - val_loss: 1.3495
```

In [97]:
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 0.8000 - loss: 1.5730
Test accuracy: 0.800000011920929
```

7. Avaliar o usa de PCA (Análise de Componentes Principais) para visualização dos dados e também como speed-up da ML (para fins de classificação).

In [98]:
```
from sklearn.decomposition import PCA

# Exemplo para visualização
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [101]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model_pca = Sequential([
    Dense(128, activation='relu'),  # Ajuste aqui
    Dense(64, activation='relu'),
    Dense(7, activation='softmax')
])

model_pca.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [103]:
```
history_pca = model_pca.fit(X_train_pca, y_train,
                            epochs=70,
                            batch_size=1,
                            validation_split=0.1)
```

```
Epoch 1/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.1514 - loss: 2.030
0 - val_accuracy: 0.2500 - val_loss: 2.6069
Epoch 2/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5251 - loss: 1.206
2 - val_accuracy: 0.2500 - val_loss: 2.1892
Epoch 3/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.3997 - loss: 1.169
5 - val_accuracy: 0.1250 - val_loss: 2.0621
Epoch 4/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5338 - loss: 1.103
9 - val_accuracy: 0.3750 - val_loss: 1.9794
Epoch 5/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4693 - loss: 1.183
1 - val_accuracy: 0.3750 - val_loss: 2.2448
Epoch 6/70
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6189 - loss: 1.044
7 - val_accuracy: 0.2500 - val_loss: 2.0866
Epoch 7/70
```

In [104]:
```
test_loss, test_acc = model_pca.evaluate(X_test_pca, y_test)
print('Test accuracy:', test_acc)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - accuracy: 0.6000 - loss: 1.6965
Test accuracy: 0.6000000238418579
```

In [107]:
```
import numpy as np
import pandas as pd
import plotly.express as px

# Supondo que y_train esteja em formato one-hot
# Converter y_train de one-hot para rótulos de classe única, se necessário
if y_train.ndim > 1 and y_train.shape[1] > 1:
    y_train_labels = np.argmax(y_train, axis=1)
else:
    y_train_labels = y_train  # Se y_train já estiver em formato de rótulos

# Criando o DataFrame
df_pca = pd.DataFrame(X_train_pca, columns=['Componente Principal 1', 'Comp
df_pca['Classe'] = y_train_labels

# Plotando com Plotly Express
fig = px.scatter(df_pca,
                 x='Componente Principal 1',
                 y='Componente Principal 2',
                 color='Classe',
                 title="Visualização dos Dados PCA",
                 labels={"Classe": "Classe"},
                 color_continuous_scale=px.colors.sequential.Viridis)

fig.show()
```

In [ ]:
```

```

8. Conclusão

obs.: Esses dados estão desatualizados, não necessariamente representam os eletrodomésticos atuais. Mas, esses estudos ainda são atuais e continuam usando recursos de IA em suas soluções.

Type *Markdown* and LaTeX: α^2