SSB CNN Modeli - 1

May 9, 2025

CNN modelleri genelde görüntü verisi veya zaman serisi gibi 2D/3D yapılarla çalışır. Senin şu anki IoT verin ise bir tablo: satırlar = örnekler, sütunlar = özellikler (yani 1D structured data). Bu tabloyu CNN'e uygun hale getirmek için "1D Convolutional" model eğiteceğiz.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,␣
↪Dropout
```

[2]:
```
# Veriyi oku
df = pd.read_csv("C:\\Users\\Lenovo\\Desktop\\OneDrive_2025-05-07\\TON_IoT␣
↪datasets\\Processed_datasets\\Processed_IoT_dataset\\IoT_Fridge.csv")

# Gereksiz sütunları çıkar
df.drop(['date', 'time', 'type'], axis=1, inplace=True)

# Kategorik veriyi dönüştür
df = pd.get_dummies(df, columns=['temp_condition'], drop_first=True)

# Giriş ve hedef
X = df.drop('label', axis=1)
y = df['label']

# Normalizasyon
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# CNN için input şekli (örnek sayısı, zaman adımı/girdi boyutu, kanal sayısı)
X_reshaped = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

# Eğitim-test böl
```

```
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.
↪2, random_state=42)
```

[5]:
```
# Model oluştur
model = Sequential([
    Conv1D(32, kernel_size=2, activation='relu', input_shape=(X_train.shape[1],␣
↪1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.1),
    Dense(1, activation='sigmoid')  # Binary output
])

# Derle
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

# Eğit
history = model.fit(X_train, y_train, epochs=30, batch_size=64,␣
↪validation_split=0.2, class_weight={0: 1, 1: 4})
```

```
Epoch 1/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4746 -
accuracy: 0.7997 - val_loss: 0.3108 - val_accuracy: 0.8007
Epoch 2/30
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4545 -
accuracy: 0.7991 - val_loss: 0.3033 - val_accuracy: 0.8010
Epoch 3/30
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4524 -
accuracy: 0.7992 - val_loss: 0.3128 - val_accuracy: 0.8007
Epoch 4/30
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4515 -
accuracy: 0.7991 - val_loss: 0.3187 - val_accuracy: 0.8007
Epoch 5/30
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4511 -
accuracy: 0.7990 - val_loss: 0.3117 - val_accuracy: 0.8010
Epoch 6/30
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4507 -
accuracy: 0.7990 - val_loss: 0.3165 - val_accuracy: 0.8007
Epoch 7/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4503 -
accuracy: 0.7991 - val_loss: 0.3190 - val_accuracy: 0.8007
Epoch 8/30
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4502 -
accuracy: 0.7990 - val_loss: 0.3198 - val_accuracy: 0.8010
Epoch 9/30
```

```
5871/5871 [==============================] - 26s 4ms/step - loss: 0.4503 -
accuracy: 0.7990 - val_loss: 0.3149 - val_accuracy: 0.8010
Epoch 10/30
5871/5871 [==============================] - 29s 5ms/step - loss: 0.4500 -
accuracy: 0.7991 - val_loss: 0.3188 - val_accuracy: 0.8007
Epoch 11/30
5871/5871 [==============================] - 24s 4ms/step - loss: 0.4500 -
accuracy: 0.7990 - val_loss: 0.3126 - val_accuracy: 0.8010
Epoch 12/30
5871/5871 [==============================] - 25s 4ms/step - loss: 0.4499 -
accuracy: 0.7991 - val_loss: 0.3183 - val_accuracy: 0.8010
Epoch 13/30
5871/5871 [==============================] - 27s 5ms/step - loss: 0.4499 -
accuracy: 0.7989 - val_loss: 0.3163 - val_accuracy: 0.8007
Epoch 14/30
5871/5871 [==============================] - 27s 5ms/step - loss: 0.4502 -
accuracy: 0.7989 - val_loss: 0.3159 - val_accuracy: 0.8007
Epoch 15/30
5871/5871 [==============================] - 27s 5ms/step - loss: 0.4496 -
accuracy: 0.7989 - val_loss: 0.3158 - val_accuracy: 0.8010
Epoch 16/30
5871/5871 [==============================] - 26s 5ms/step - loss: 0.4495 -
accuracy: 0.7990 - val_loss: 0.3137 - val_accuracy: 0.8007
Epoch 17/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4499 -
accuracy: 0.7989 - val_loss: 0.3146 - val_accuracy: 0.8007
Epoch 18/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4500 -
accuracy: 0.7988 - val_loss: 0.3182 - val_accuracy: 0.8010
Epoch 19/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4495 -
accuracy: 0.7990 - val_loss: 0.3195 - val_accuracy: 0.8007
Epoch 20/30
5871/5871 [==============================] - 27s 5ms/step - loss: 0.4494 -
accuracy: 0.7991 - val_loss: 0.3163 - val_accuracy: 0.8010
Epoch 21/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4498 -
accuracy: 0.7991 - val_loss: 0.3183 - val_accuracy: 0.8007
Epoch 22/30
5871/5871 [==============================] - 29s 5ms/step - loss: 0.4491 -
accuracy: 0.7992 - val_loss: 0.3153 - val_accuracy: 0.8010
Epoch 23/30
5871/5871 [==============================] - 29s 5ms/step - loss: 0.4498 -
accuracy: 0.7991 - val_loss: 0.3143 - val_accuracy: 0.8007
Epoch 24/30
5871/5871 [==============================] - 30s 5ms/step - loss: 0.4496 -
accuracy: 0.7993 - val_loss: 0.3168 - val_accuracy: 0.8007
Epoch 25/30
```

```
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4490 -
accuracy: 0.7992 - val_loss: 0.3157 - val_accuracy: 0.8007
Epoch 26/30
5871/5871 [==============================] - 30s 5ms/step - loss: 0.4490 -
accuracy: 0.7992 - val_loss: 0.3167 - val_accuracy: 0.8007
Epoch 27/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4488 -
accuracy: 0.7992 - val_loss: 0.3139 - val_accuracy: 0.8007
Epoch 28/30
5871/5871 [==============================] - 29s 5ms/step - loss: 0.4492 -
accuracy: 0.7993 - val_loss: 0.3150 - val_accuracy: 0.8007
Epoch 29/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4489 -
accuracy: 0.7992 - val_loss: 0.3155 - val_accuracy: 0.8007
Epoch 30/30
5871/5871 [==============================] - 28s 5ms/step - loss: 0.4486 -
accuracy: 0.7993 - val_loss: 0.3177 - val_accuracy: 0.8007
```

[6]:
```python
# Tahmin yap
y_pred = model.predict(X_test)
y_pred = [1 if i > 0.5 else 0 for i in y_pred]

# Raporla
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
3670/3670 [==============================] - 8s 2ms/step
Confusion Matrix:
[[76619 23408]
 [    0 17389]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87    100027
           1       0.43      1.00      0.60     17389

    accuracy                           0.80    117416
   macro avg       0.71      0.88      0.73    117416
weighted avg       0.92      0.80      0.83    117416
```

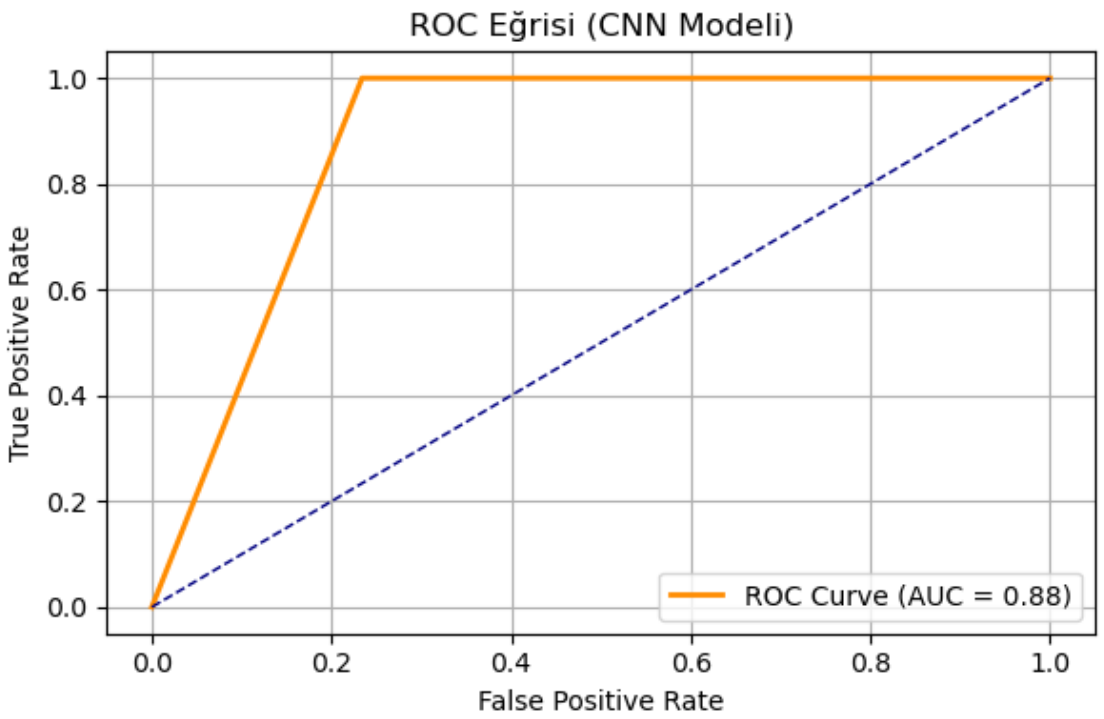
Modeli eğitirken başta Epoch sayısını 10, dropoutu 0.1 den 0.3 e çektim ve class_weight={0:1, 1:4} ekledim. Böylece -class_weight={0:1, 1:4} — anomali verisini değerli hale getirdin -epochs=30 — model daha uzun öğrenme şansı buldu -Dropout oranı optimize edildi — bilgi kaybı azaltıldı -CNN katmanları doğru yapılandırıldı — örüntüleri daha iyi öğrendi

Anomali tespiti Recall (1 için): 1.00 → Hiçbir saldırıyı kaçırmamışsın Precision (1 için): 0.43 → Yanlış alarmlar hâlâ var ama bu çok normal.

F1-Score: Normal: 0.87 Anomali: 0.60 Makro Ortalama: 0.73 Bu, modelin her iki sınıfı da oldukça dengeli tanıdığını gösteriyor.

[7]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

#  Modelin decision output'u (olasılıklar) ile tahminler (0-1 arası)
y_proba = model.predict(X_test)

#  ROC eğrisi için fpr (False Positive Rate) ve tpr (True Positive Rate) hesapla
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

#  AUC (Area Under Curve) skoru
roc_auc = auc(fpr, tpr)

#  ROC eğrisini çiz
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')  # referans eğrisi
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Eğrisi (CNN Modeli)')
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
3670/3670 [==============================] - 8s 2ms/step
```



y_proba: model.predict() sonucu 0-1 arası olasılıklardır. ROC için bunlar gerekir. fpr ve tpr: her eşik değeri için ne kadar doğru/yanlış tahmin yapıldığını gösterir. AUC: eğrinin altındaki alan, 1.00'a ne kadar yakınsa o kadar iyi.

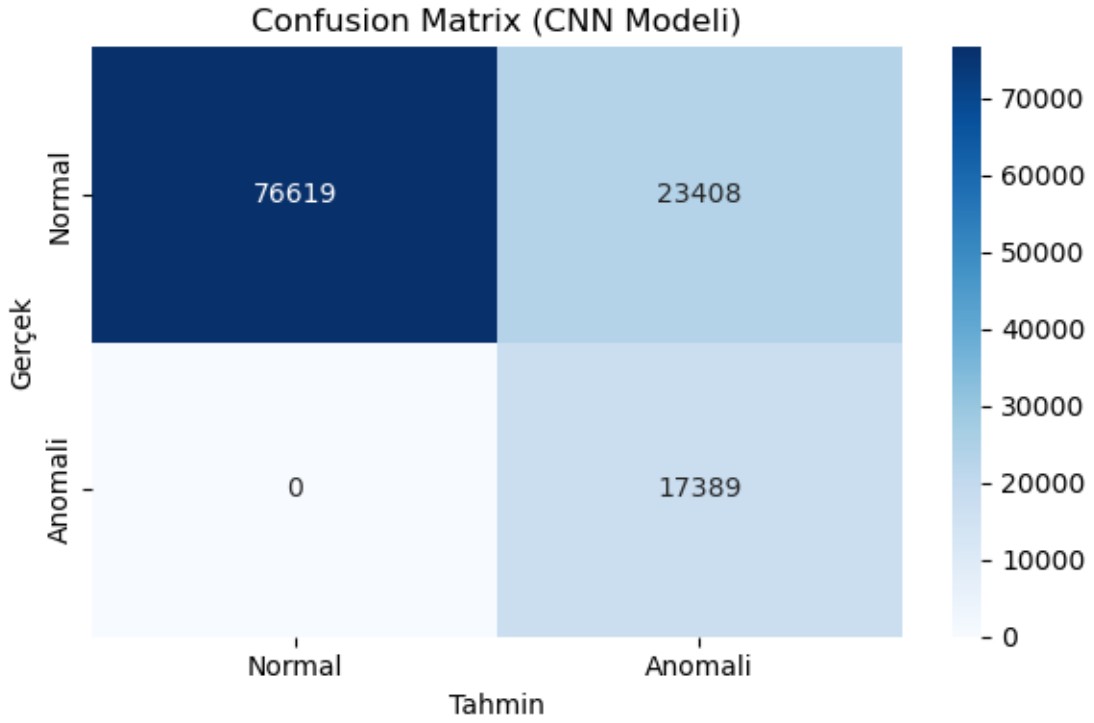
ROC Eğrisi X ekseni: False Positive Rate (Yanlış alarm oranı) Y ekseni: True Positive Rate (Doğru pozitif oranı → yani anomaliyi yakalama gücü)

AUC 0.88 Bu skor [0.50 - 1.00] aralığındadır. 0.50 = rastgele tahmin. 1.00 = mükemmel tahmin. Benim sonucum 0.88

CNN model: Anomalileri yüksek doğrulukla ayırt edebiliyor. Sistemi güvenli hale getirmek için oldukça uygun

[8]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#  Confusion matrix hesapla
cm = confusion_matrix(y_test, y_pred)

#  Görselleştir
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Normal", "Anomali"],
            yticklabels=["Normal", "Anomali"])
```

```
plt.title("Confusion Matrix (CNN Modeli)")
plt.xlabel("Tahmin")
plt.ylabel("Gerçek")
plt.tight_layout()
plt.show()
```



annot=True: her hücreye sayıları yazdırır. fmt="d": sayıları tam sayı olarak formatlar. cmap="Blues": mavi tonuyla boyar. xticklabels ve yticklabels: eksen adlarını senin verine göre ayarladık.

Doğru Pozitif (TP): 17.389 Model gerçekten anomali olan tüm verileri doğru tespit etmiş. Recall (1 sınıfı için) = 1.00 → Hiçbir saldırıyı kaçırmamış.

False Positive (FP): 23.408 23.408 tane normal davranışı "anomali" zannetmiş. Precision (1 için): 0.43 Bu oranı iyileştirmek mümkün ama anomali kaçırmama pahasına bazı yanlış alarmlar kabul edilebilir. Yani: güvenlik öncelikli modellerde bu feda edilebilir

Doğru Negatif (TN): 76.619 Normal olan davranışları doğru tanımış. Bu da True Negative Rate'in iyi olduğunu gösteriyor.

Genel Sonuç: Anomaliyi asla kaçırmıyor (1.00 recall) Biraz fazla alarm veriyor ama bu güvenlik için kabul edilebilir Modelin savunmaya geçme refleksi çok güçlü Confusion Matrix'teki görselleştirme farkları (mavi tonlar) sınıflar arası ayrımı çok net gösteriyor

Model Yapısı:Conv1D → MaxPooling1D → Dropout → Flatten → Dense → Output

Eğitim Sonuçları: Validation Accuracy hep 0.80 civarında stabil kalmış. Model overfitting yapmamış. Loss değerleri 30 epoch boyunca düşmüş veya sabit kalmış → Model öğreniyor.

Classification Report: Recall (1) = 1.00 → Hiç saldırı kaçırmamışsın. Precision (1) = 0.43 → Yanlış alarmları daha sonra filtreleyebilirsin. F1-score (anomali) = 0.60 → Dengeli başarı.

[ ]: