Modelo Final Notebook

```
# MODELO FINAL OPTIMIZADO

import numpy as np
import pandas as pd
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix
)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
import json, os

# ------------------------------------------------------------------------
# 1) Generar dataset sintético
# ------------------------------------------------------------------------
def generate_synthetic_dataset(n_samples=5000, fall_ratio=0.12, random_state=42):
    rng = np.random.default_rng(random_state)
    n_falls = int(n_samples * fall_ratio)
    n_non = n_samples - n_falls

    rows = []
    axes = ['ax','ay','az','gx','gy','gz']
    stats = ['mean','std','rms','energy']

    # No Caídas
    for i in range(n_non):
        feats=[]
        for ax in axes:
            mean = rng.normal(0,0.2)
            std = abs(rng.normal(0.2,0.05))
            rms = np.sqrt(mean**2 + std**2)
            energy = abs(rng.normal(0.5,0.2))
            feats.extend([mean,std,rms,energy])
        rows.append(feats + [0])

    # Caídas
    for i in range(n_falls):
        feats=[]
        for ax in axes:
            mean = rng.normal(0,0.6)
            std = abs(rng.normal(0.8,0.2))
            rms = np.sqrt(mean**2 + std**2)
            energy = abs(rng.normal(2.5,0.8))
            feats.extend([mean,std,rms,energy])
        rows.append(feats + [1])

    data = np.array(rows)
    X = data[:, :-1].astype(float).reshape(-1, 4, 6, 1)
    y = data[:, -1].astype(int)

    return X, y

print("Generando dataset final...")
X, y = generate_synthetic_dataset()

# Convertir etiquetas a one-hot (para Keras)
y_cat = to_categorical(y, num_classes=2)
# ------------------------------------------------------------------------
```

```
# 2) Separación en train / test
# ------------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y_cat, test_size=0.2, random_state=42, stratify=y
)

# ------------------------------------------------------------------
# 3) Construcción del modelo optimizado
# ------------------------------------------------------------------
def build_cnn_model():
    model = Sequential([
        Conv2D(16, (2,2), activation='relu', padding='same', input_shape=(4,6,1)),
        MaxPooling2D((2,2)),
        Conv2D(32, (2,2), activation='relu', padding='same'),
        Flatten(),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dense(2, activation='softmax')
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

# ------------------------------------------------------------------
# 4) Entrenamiento del modelo
# ------------------------------------------------------------------
print("\nEntrenando modelo optimizado final...")
model = build_cnn_model()
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)

# ------------------------------------------------------------------
# 5) Evaluación completa (Semana 7)
# ------------------------------------------------------------------
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)

acc  = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred)
rec  = recall_score(y_true, y_pred)
f1   = f1_score(y_true, y_pred)
cm   = confusion_matrix(y_true, y_pred)

spec = cm[0,0] / (cm[0,0] + cm[0,1])  # Specificity

print("\n===== MÉTRICAS FINALES =====")
print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", rec)
print("F1-score:", f1)
print("Specificity:", spec)
print("\nMatriz de confusión:")
print(cm)

# ------------------------------------------------------------------
# 6) Gráfica de matriz de confusión
# ------------------------------------------------------------------
plt.figure(figsize=(4,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Matriz de Confusión - Modelo Final Optimizado")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()

# ------------------------------------------------------------------
# 7) Exportación del modelo
# ------------------------------------------------------------------
os.makedirs("modelo_final", exist_ok=True)
```

```
model.save("modelo_final/modelo_final_optimo.h5")
model.save("modelo_final/modelo_final_optimo.keras")

print("\nModelo exportado correctamente.")

# ------------------------------------------------------------------------
# 8) Salida JSON de ejemplo
# ------------------------------------------------------------------------
sample = np.random.normal(0, 0.4, (1,4,6,1))

pred = model.predict(sample)
label = np.argmax(pred)
confidence = float(np.max(pred))

output = {
    "prediccion": "caída" if label == 1 else "no_caída",
    "probabilidad": confidence
}

with open("modelo_final/salida_ejemplo.json", "w", encoding="utf-8") as f:
    json.dump(output, f, indent=4, ensure_ascii=False)

print("\nSalida JSON generada:")
print(json.dumps(output, indent=4, ensure_ascii=False))

print("\n=== MODELO FINAL ===")
```