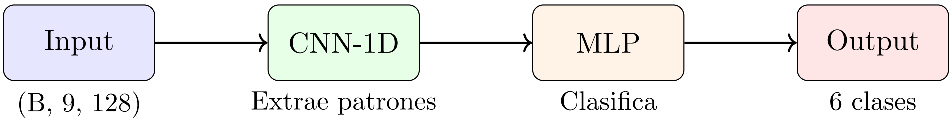
Clasificación de Actividades Humanas

Red Neuronal Híbrida CNN-1D + MLP



Guía de Estudio Personal

Conceptos Matemáticos, Arquitectura y Análisis de Resultados

Resultado Final

95.64 % Accuracy en Test

Proyecto HAR - Human Activity Recognition

Índice

1. Introducción y Objetivo **3**
1.1. ¿Qué es HAR? 3
1.2. ¿Por qué CNN-1D + MLP? 3
1.3. Resumen de Resultados 3

2. Los Datos: Entendiendo el Input **3**
2.1. Estructura del Dataset 3
2.2. Los 9 Canales de Sensores 4
2.3. Distribución de Clases 4
2.4. Visualización de Señales 5

3. Conceptos Matemáticos Clave **5**
3.1. Tensores: La Estructura de los Datos 5
3.2. ¿Qué es un Kernel (Filtro)? 5
3.3. Convolución 1D: La Operación Matemática 6
3.4. Padding: Mantener el Tamaño 7
3.5. Batch Normalization: Estandarización Dinámica 7
3.6. MaxPooling: Reducción de Dimensionalidad 8
3.7. ReLU: Introduciendo No-Linealidad 8
3.8. Dropout: Regularización por Exclusión 8
3.9. Cross-Entropy Loss: Midiendo el Error 9

4. Arquitectura del Modelo **10**
4.1. Diagrama Completo 10
4.2. Flujo de Dimensiones (Con Números Reales) 10
4.3. Cálculo de Parámetros 11
4.4. Código de la Arquitectura 11

5. Entrenamiento **12**
5.1. Configuración de Hiperparámetros 12
5.2. El Loop de Entrenamiento 12
5.3. Backpropagation: La Regla de la Cadena 13
5.4. AdamW: El Optimizador 13
5.5. Early Stopping: Evitando Overfitting 13
5.6. Curvas de Entrenamiento 14

6. Resultados y Análisis **14**
6.1. Métricas Globales 14
6.2. Métricas por Clase 14
6.3. Matriz de Confusión 15
6.4. Análisis de Confusiones 15
6.5. Gráfico de Métricas por Clase 16
6.6. Resumen de Observaciones 16

7. Preguntas Potenciales del Profesor **17**
7.1. Conceptos Fundamentales 17
7.2. Arquitectura 18
7.3. Entrenamiento 19
7.4. Resultados 19
7.5. Matemáticas 20

8. Conclusiones **21**
8.1. Logros del Proyecto 21
8.2. Conceptos Matemáticos Aplicados 21
8.3. Posibles Mejoras 21

1. Introducción y Objetivo

1.1. ¿Qué es HAR?

Human Activity Recognition (HAR) es la tarea de clasificar automáticamente qué actividad física está realizando una persona, usando datos de sensores inerciales (acelerómetro y giroscopio) de un smartphone.

Objetivo del Proyecto

Construir un modelo de deep learning que clasifique correctamente 6 actividades humanas:

1. **WALKING** - Caminando
2. **WALKING_UPSTAIRS** - Subiendo escaleras
3. **WALKING_DOWNSTAIRS** - Bajando escaleras
4. **SITTING** - Sentado
5. **STANDING** - De pie
6. **LAYING** - Acostado

1.2. ¿Por qué CNN-1D + MLP?

- **CNN-1D**: Las señales de sensores son series temporales. La convolución 1D es perfecta para detectar patrones locales en secuencias (picos, ciclos, transiciones).
- **MLP**: Después de extraer features con CNN, usamos capas fully-connected para combinarlas y tomar la decisión final de clasificación.

1.3. Resumen de Resultados

Métrica	Valor
Accuracy en Test	**95.64 %**
Accuracy en Validación	97.82 %
Épocas entrenadas	73 (early stopping en 88)
Total de parámetros	1,095,558

2. Los Datos: Entendiendo el Input

2.1. Estructura del Dataset

El dataset contiene señales de sensores capturadas a 50Hz durante 2.56 segundos (ventanas de 128 muestras).

Dimensiones del Tensor de Entrada

$$X \in \mathbb{R}^{N \times C \times T}$$

Donde:

- N = número de muestras (7,352 train, 2,947 test)
- C = 9 canales de sensores
- T = 128 timesteps (pasos temporales)

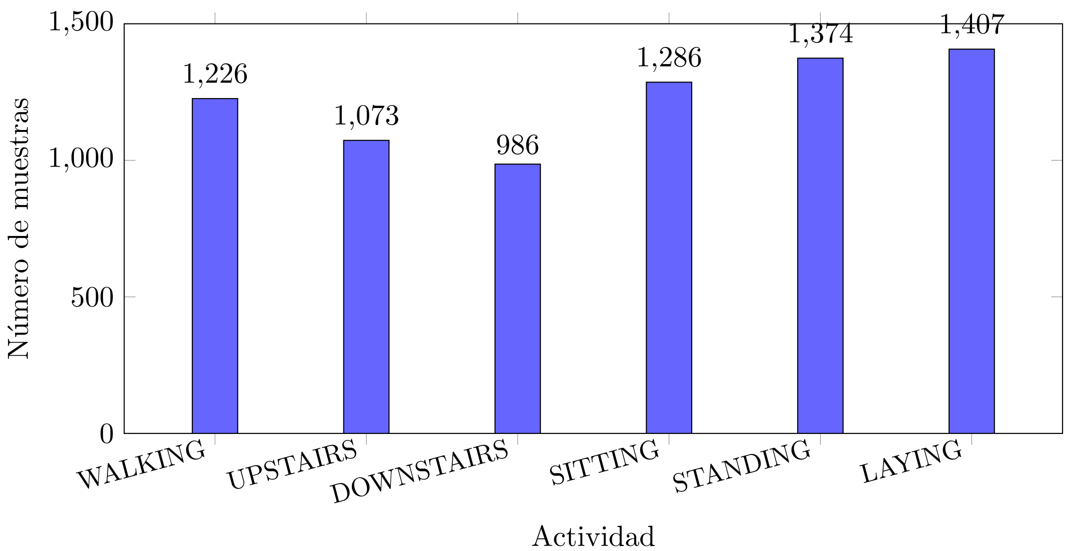
2.2. Los 9 Canales de Sensores

Canal	Nombre	Media (μ)	Std (σ)
0	body_acc_x	0.0001	0.1412
1	body_acc_y	-0.0015	0.1024
2	body_acc_z	0.0003	0.0961
3	body_gyro_x	-0.0004	0.2870
4	body_gyro_y	0.0005	0.2197
5	body_gyro_z	0.0002	0.1917
6	total_acc_x	-0.3214	0.2012
7	total_acc_y	0.0824	0.1501
8	total_acc_z	0.0901	0.1284

Interpretación Física

- **body_acc**: Aceleración del cuerpo (sin gravedad)
- **body_gyro**: Velocidad angular (rotación)
- **total_acc**: Aceleración total (incluye gravedad)
- **x, y, z**: Los tres ejes espaciales

2.3. Distribución de Clases

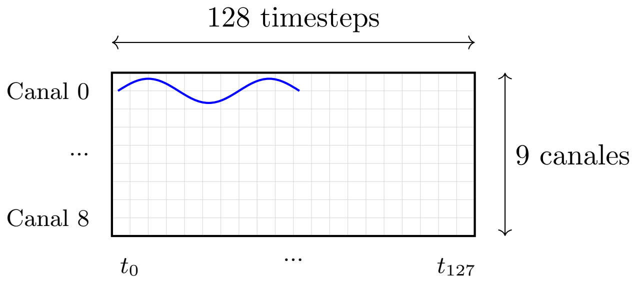


Observación Importante

Las clases están **relativamente balanceadas** (16-19 % cada una). Esto significa que el accuracy es una métrica confiable. Si hubiera desbalance severo (ej: 90 % vs 10 %), necesitaríamos métricas adicionales como F1-Score ponderado.

2.4. Visualización de Señales

Cada muestra es una **matriz de 9 filas × 128 columnas**. Visualmente:



3. Conceptos Matemáticos Clave

3.1. Tensores: La Estructura de los Datos

Definición de Tensor

Un **tensor** es una generalización de matrices a múltiples dimensiones:

- **Orden 0**: Escalar (un número) → 42
- **Orden 1**: Vector → $[1, 2, 3]$
- **Orden 2**: Matriz → $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
- **Orden 3**: Tensor 3D → Nuestros datos (N, 9, 128)

Ejemplo con nuestros datos:

- $X[0]$ = Primera muestra (matriz 9×128)
- $X[0, 3]$ = Canal 3 de la primera muestra (vector de 128)
- $X[0, 3, 50]$ = Valor en el timestep 50, canal 3, muestra 0 (escalar)

3.2. ¿Qué es un Kernel (Filtro)?

Definición de Kernel

Un **kernel** (también llamado **filtro**) es un pequeño vector de pesos que se "desliza" sobre la señal para detectar patrones locales.
En nuestro modelo usamos `kernel_size=5`, lo que significa que cada kernel mira 5 valores consecutivos de la señal.

Visualización del Kernel:

Señal: | x_0 | x_1 | x_2 | x_3 | x_4 | x_5 | $\cdots$

Kernel: | w_0 | w_1 | w_2 | w_3 | w_4 | $\longrightarrow y_0 = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$

$\xrightarrow{}$ desliza

¿Qué detecta un kernel?
Diferentes valores de pesos detectan diferentes patrones:

- $[1, 0, 0, 0, -1] \to$ Detecta cambios (derivada discreta)
- $[0.2, 0.2, 0.2, 0.2, 0.2] \to$ Suaviza la señal (promedio móvil)
- $[-1, 2, -1, 2, -1] \to$ Detecta oscilaciones rápidas

Clave

Los valores del kernel **se aprenden automáticamente** durante el entrenamiento. El modelo descubre qué patrones son útiles para clasificar cada actividad.

3.3. Convolución 1D: La Operación Matemática

Fórmula de Convolución 1D

$$y[t] = \sum_{k=0}^{K-1} w[k] \cdot x[t+k] + b$$

Donde:

- x = señal de entrada (128 valores)
- w = kernel/filtro (5 valores en nuestro caso)
- K = tamaño del kernel (5)
- b = sesgo (bias)
- y = señal de salida

Ejemplo numérico con nuestros datos:
Supongamos una señal de acelerómetro y un kernel entrenado:

$$\begin{aligned} x &= [0.12, 0.45, 0.78, 0.56, 0.23, 0.67, ...] \\ w &= [0.3, -0.5, 0.2, -0.4, 0.6] \quad \text{(valores aprendidos)} \\ b &= 0.1 \end{aligned}$$

Calculamos la primera salida ($t = 0$):

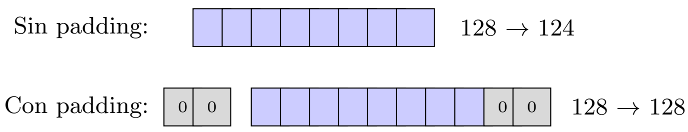
$$\begin{aligned} y[0] &= (0.3)(0.12) + (-0.5)(0.45) + (0.2)(0.78) + (-0.4)(0.56) + (0.6)(0.23) + 0.1 \\ &= 0.036 - 0.225 + 0.156 - 0.224 + 0.138 + 0.1 \\ &= \boxed{-0.019} \end{aligned}$$

3.4. Padding: Mantener el Tamaño

¿Qué es Padding?

El **padding** agrega ceros al inicio y final de la señal para que la salida tenga el mismo tamaño que la entrada.
Con `padding=2` y `kernel_size=5`:

$$\text{Output size} = \text{Input size} - \text{kernel\_size} + 1 + 2 \times \text{padding} = 128 - 5 + 1 + 4 = 128$$



3.5. Batch Normalization: Estandarización Dinámica

Fórmula de Batch Normalization

Para cada batch de datos, se aplica la estandarización (z-score):

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Donde:

- μ_B = media del batch
- σ_B^2 = varianza del batch
- ϵ = constante pequeña para evitar división por cero (10^{-5})

Luego se aplican parámetros aprendibles:

$$y_i = \gamma \hat{x}_i + \beta$$

Ejemplo numérico:
Supongamos un mini-batch con valores $[2.0, 4.0, 6.0, 8.0]$:

$$\mu_B = \frac{2+4+6+8}{4} = 5.0$$
$$\sigma_B^2 = \frac{(2-5)^2 + (4-5)^2 + (6-5)^2 + (8-5)^2}{4} = \frac{9+1+1+9}{4} = 5.0$$
$$\hat{x}_1 = \frac{2.0 - 5.0}{\sqrt{5.0 + 10^{-5}}} = \frac{-3.0}{2.236} = \boxed{-1.342}$$

¿Por qué usar BatchNorm?

1. **Estabiliza el entrenamiento**: Evita que las activaciones crezcan o se reduzcan demasiado
2. **Permite learning rates más altos**: El modelo converge más rápido
3. **Actúa como regularizador**: Reduce ligeramente el overfitting

3.6. MaxPooling: Reducción de Dimensionalidad

Fórmula de Max Pooling 1D

Con `kernel_size=2`:

$$y[i] = \max(x[2i], x[2i+1])$$

Reduce la dimensión temporal a la mitad, conservando los valores más importantes.

Ejemplo:

Input: | 3 | 7 | 2 | 9 | 5 | 1 | 8 | 4 | 8 valores

$\downarrow$ max()

Output: | 7 | 9 | 5 | 8 | 4 valores

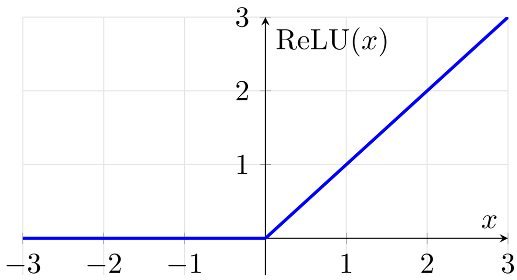
En nuestro modelo:

- Después de Conv1 + Pool: 128 → 64
- Después de Conv2 + Pool: 64 → 32

3.7. ReLU: Introduciendo No-Linealidad

Función ReLU

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$



¿Por qué es necesaria?

Sin funciones de activación no-lineales, apilar capas lineales sería equivalente a una sola capa lineal:

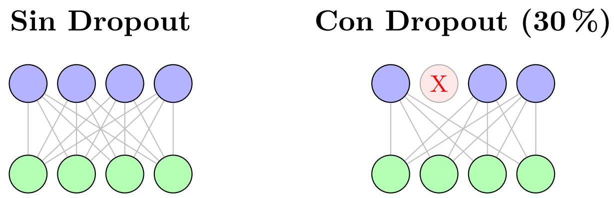
$$W_2 \cdot (W_1 \cdot x) = (W_2 W_1) \cdot x = W_{equiv} \cdot x$$

ReLU permite aprender funciones complejas y no-lineales.

3.8. Dropout: Regularización por Exclusión

¿Qué hace Dropout?

Durante el entrenamiento, **apaga aleatoriamente** neuronas con probabilidad p (en nuestro caso, $p = 0.3$ o $30\,\%$).
Esto obliga a la red a no depender demasiado de ninguna neurona individual, mejorando la generalización.



3.9. Cross-Entropy Loss: Midiendo el Error

Fórmula de Cross-Entropy

Para una muestra con etiqueta real y (one-hot) y predicción $\hat{p}$ (probabilidades):

$$L = -\sum_{c=1}^{C} y_c \cdot \log(\hat{p}_c)$$

Como y es one-hot (solo un 1, resto 0), se simplifica a:

$$L = -\log(\hat{p}_{clase\_correcta})$$

Ejemplo numérico:
Si el modelo predice para una muestra de WALKING (clase 0):

$$\hat{p} = [0.85, 0.05, 0.03, 0.02, 0.03, 0.02]$$

El loss es:

$$L = -\log(0.85) = -(-0.163) = \boxed{0.163}$$

Si la predicción fuera más incierta, $\hat{p}_0 = 0.30$:

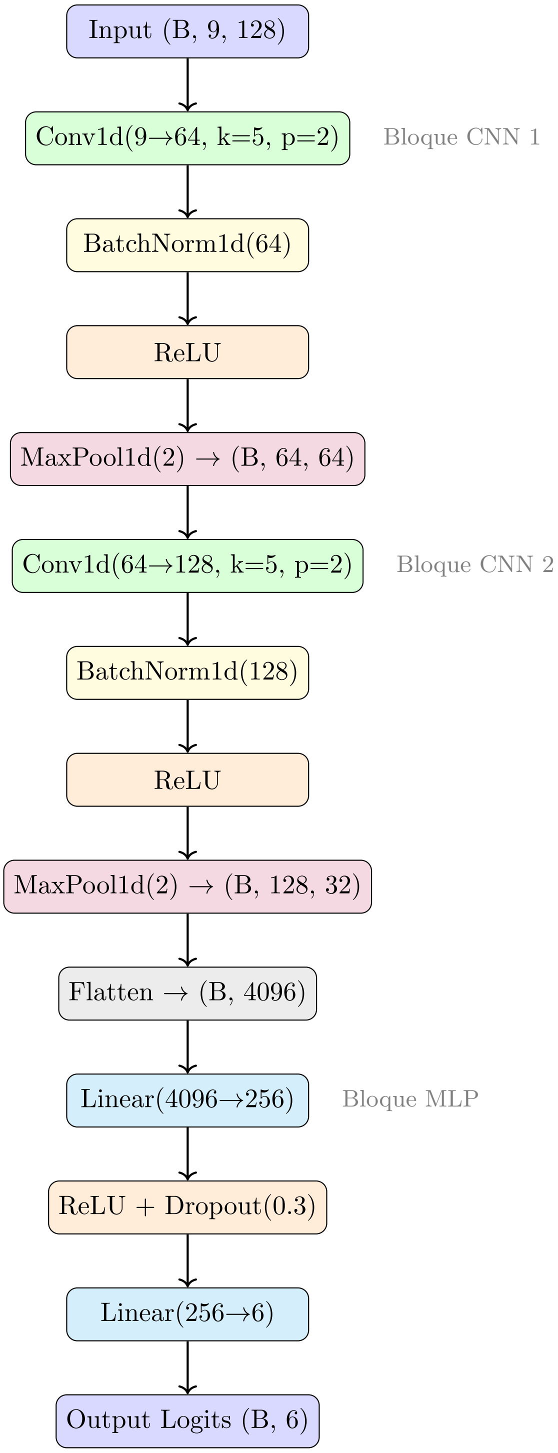
$$L = -\log(0.30) = \boxed{1.204}$$

Interpretación

- Loss **bajo** (≈ 0): El modelo está muy seguro de la clase correcta
- Loss **alto** (> 1): El modelo está confundido o equivocado
- Loss $\to \infty$: Si $\hat{p}_{correcta} \to 0$

4. Arquitectura del Modelo

4.1. Diagrama Completo



4.2. Flujo de Dimensiones (Con Números Reales)

Siguiendo una muestra a través de la red (batch size = 64):

Capa	Operación	Shape de Salida
Input	—	(64, 9, 128)
Conv1d	64 filtros, kernel=5, padding=2	(64, 64, 128)
BatchNorm1d	Normaliza 64 canales	(64, 64, 128)
ReLU	max(0, x)	(64, 64, 128)
MaxPool1d	kernel=2	(64, 64, **64**)
Conv1d	128 filtros, kernel=5, padding=2	(64, 128, 64)
BatchNorm1d	Normaliza 128 canales	(64, 128, 64)
ReLU	max(0, x)	(64, 128, 64)
MaxPool1d	kernel=2	(64, 128, **32**)
Flatten	Aplana	(64, **4096**)
Linear	4096 → 256	(64, 256)
ReLU	max(0, x)	(64, 256)
Dropout	30 % apagado	(64, 256)
Linear	256 → 6	(64, **6**)

4.3. Cálculo de Parámetros

Fórmula para Parámetros de Conv1d

$$\text{Params} = (\text{kernel\_size} \times \text{in\_channels} + 1) \times \text{out\_channels}$$

El +1 es por el bias.

Cálculo detallado:

Capa	Fórmula	Cálculo	Parámetros
Conv1	$(5 \times 9 + 1) \times 64$	$(45 + 1) \times 64$	2,944
BatchNorm1	2×64	γ, β	128
Conv2	$(5 \times 64 + 1) \times 128$	$(320 + 1) \times 128$	41,088
BatchNorm2	2×128	γ, β	256
Linear1	$(4096 + 1) \times 256$	4097×256	1,048,832
Linear2	$(256 + 1) \times 6$	257×6	1,542
Total			**1,094,790**

Observación

El 95 % de los parámetros están en `Linear1` (la capa que conecta CNN con MLP). Esto es típico: las capas fully-connected son "costosas" en parámetros.

4.4. Código de la Arquitectura

```python
class HARModel(nn.Module):
    def __init__(self, n_channels=9, n_classes=6):
        super(HARModel, self).__init__()

        # Bloque CNN-1D
        self.conv1 = nn.Conv1d(n_channels, 64, kernel_size=5, padding=2)
        self.bn1 = nn.BatchNorm1d(64)
        self.conv2 = nn.Conv1d(64, 128, kernel_size=5, padding=2)
        self.bn2 = nn.BatchNorm1d(128)
```

```
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.dropout = nn.Dropout(0.3)

        # Bloque MLP: 128 canales * 32 timesteps = 4096
        self.fc1 = nn.Linear(128 * 32, 256)
        self.fc2 = nn.Linear(256, n_classes)

    def forward(self, x):
        # x: (B, 9, 128)
        x = self.pool(F.relu(self.bn1(self.conv1(x))))  # (B, 64, 64)
        x = self.pool(F.relu(self.bn2(self.conv2(x))))  # (B, 128, 32)
        x = x.view(x.size(0), -1)                        # (B, 4096)
        x = self.dropout(F.relu(self.fc1(x)))            # (B, 256)
        x = self.fc2(x)                                  # (B, 6)
        return x
```

5. Entrenamiento

5.1. Configuración de Hiperparámetros

Hiperparámetro	Valor
Learning rate	10^{-3} (0.001)
Weight decay (L2)	10^{-2} (0.01)
Batch size	64
Épocas máximas	100
Early stopping patience	15
Dropout rate	0.3 (30 %)
Optimizer	AdamW
Loss function	CrossEntropyLoss
Train/Val split	80 % / 20 %

5.2. El Loop de Entrenamiento

```
for epoch in range(num_epochs):
    # 1. FORWARD PASS
    outputs = model(X_batch)              # Predicciones
    loss = criterion(outputs, y_batch)  # Calcular error

    # 2. BACKWARD PASS (Backpropagation)
    optimizer.zero_grad()  # Limpiar gradientes anteriores
    loss.backward()        # Calcular gradientes (regla de la cadena)
    optimizer.step()       # Actualizar pesos: W <- W - lr * grad

    # 3. VALIDACION
    val_loss, val_acc = validate(model, val_loader)

    # 4. EARLY STOPPING
    if val_loss no mejora en 15 epocas:
        break
```

5.3. Backpropagation: La Regla de la Cadena

Regla de la Cadena

Para calcular cómo cada peso afecta el loss:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial f_{n-1}} \cdot \ldots \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_1}$$

PyTorch calcula esto automáticamente con `loss.backward()`.

Ejemplo simplificado:
Si tenemos: $y = \text{ReLU}(Wx + b)$ y $L = (y - t)^2$

$$\begin{aligned}\frac{\partial L}{\partial W} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial (Wx+b)} \cdot \frac{\partial (Wx+b)}{\partial W} \\ &= 2(y-t) \cdot \mathbf{1}_{Wx+b>0} \cdot x\end{aligned}$$

Donde $\mathbf{1}_{Wx+b>0}$ es 1 si $Wx + b > 0$ (derivada de ReLU).

5.4. AdamW: El Optimizador

Actualización de Pesos con Adam

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Donde:

- $\hat{m}_t$ = media móvil del gradiente (momento)
- $\hat{v}_t$ = media móvil del gradiente al cuadrado
- η = learning rate (10^{-3})
- ϵ = estabilidad numérica (10^{-8})

AdamW añade **weight decay** directamente a los pesos:

$$\theta_{t+1} = (1 - \lambda)\theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

con $\lambda = 10^{-2}$ (penaliza pesos grandes).

5.5. Early Stopping: Evitando Overfitting

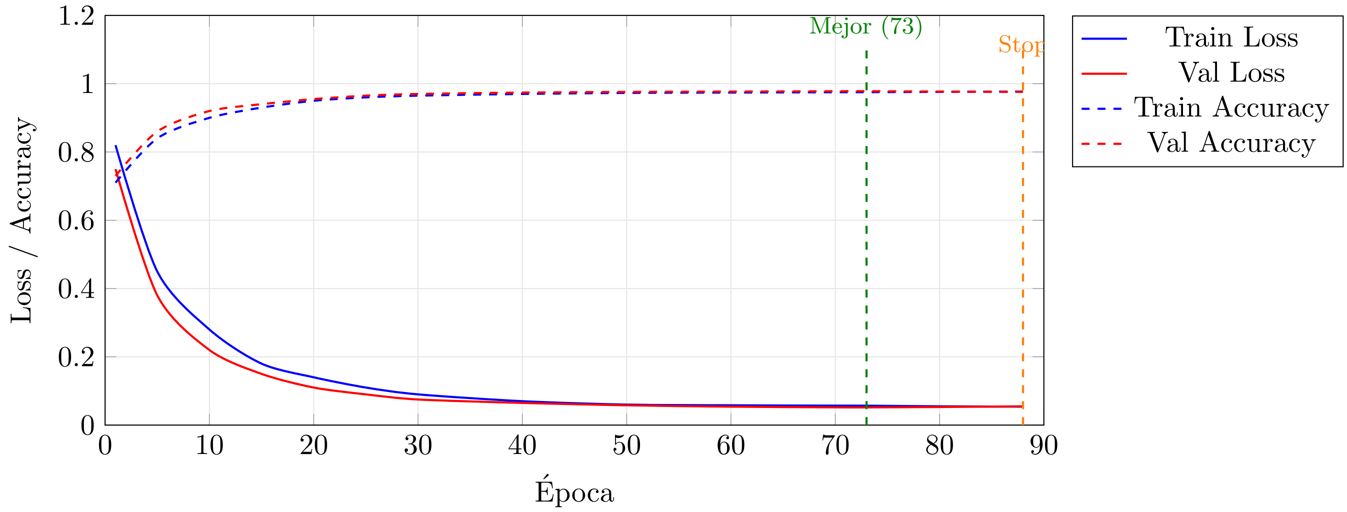
¿Cómo funciona?

1. En cada época, medir el `val_loss`
2. Si mejora: guardar el modelo
3. Si NO mejora en 15 épocas consecutivas: parar
4. Restaurar el mejor modelo guardado

En nuestro entrenamiento:

- Mejor época: 73
- Early stopping activado: época 88
- Mejor val_loss: 0.0519

5.6. Curvas de Entrenamiento



Observaciones:

- Las curvas de train y val están muy cercanas → **buena generalización**
- No hay divergencia significativa → **no hay overfitting severo**
- La mejora se estabiliza después de época 60 → el modelo convergió

6. Resultados y Análisis

6.1. Métricas Globales

Test Accuracy: 95.64 %

2,819 clasificaciones correctas de 2,947 muestras

6.2. Métricas por Clase

Actividad	Precision	Recall	F1-Score	Muestras
WALKING	1.0000	0.9496	0.9741	496
WALKING_UPSTAIRS	0.9891	0.9618	0.9752	471
WALKING_DOWNSTAIRS	0.9130	**1.0000**	0.9545	420
SITTING	0.8372	0.8167	0.8268	491
STANDING	0.8492	0.8571	0.8531	532
LAYING	0.9908	**1.0000**	0.9954	537
Promedio	**0.9299**	**0.9309**	**0.9299**	2947

Interpretación de Métricas

Precision: De todas las veces que predije clase X, ¿cuántas eran realmente X?

$$\text{Precision}_{\text{WALKING}} = \frac{TP}{TP + FP} = \frac{471}{471 + 0} = 1.0$$

Recall: De todas las muestras reales de clase X, ¿cuántas detecté?

$$\text{Recall}_{\text{WALKING}} = \frac{TP}{TP + FN} = \frac{471}{471 + 25} = 0.9496$$

F1-Score: Media armónica de Precision y Recall

$$F1 = 2 \cdot \frac{1.0 \times 0.9496}{1.0 + 0.9496} = 0.9741$$

6.3. Matriz de Confusión

La matriz de confusión muestra las predicciones vs las etiquetas reales:

	WALK	**UP**	**DOWN**	**SIT**	**STAND**	**LAY**
WALK	green!40471	0	24	0	1	0
UP	0	green!40453	16	0	2	0
DOWN	0	0	green!40420	0	0	0
SIT	0	0	0	green!30401	red!3081	5
STAND	0	5	0	red!3076	green!30456	0
LAY	0	0	0	2	0	green!40537

Lectura: Filas = clase real, Columnas = clase predicha. La diagonal (verde) son aciertos.

6.4. Análisis de Confusiones

Real	**Predicho**	**Veces**
SITTING	STANDING	81
STANDING	SITTING	76
WALKING	WALKING_DOWNSTAIRS	24
WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	16
STANDING	WALKING_UPSTAIRS	5

¿Por qué SITTING y STANDING se confunden?

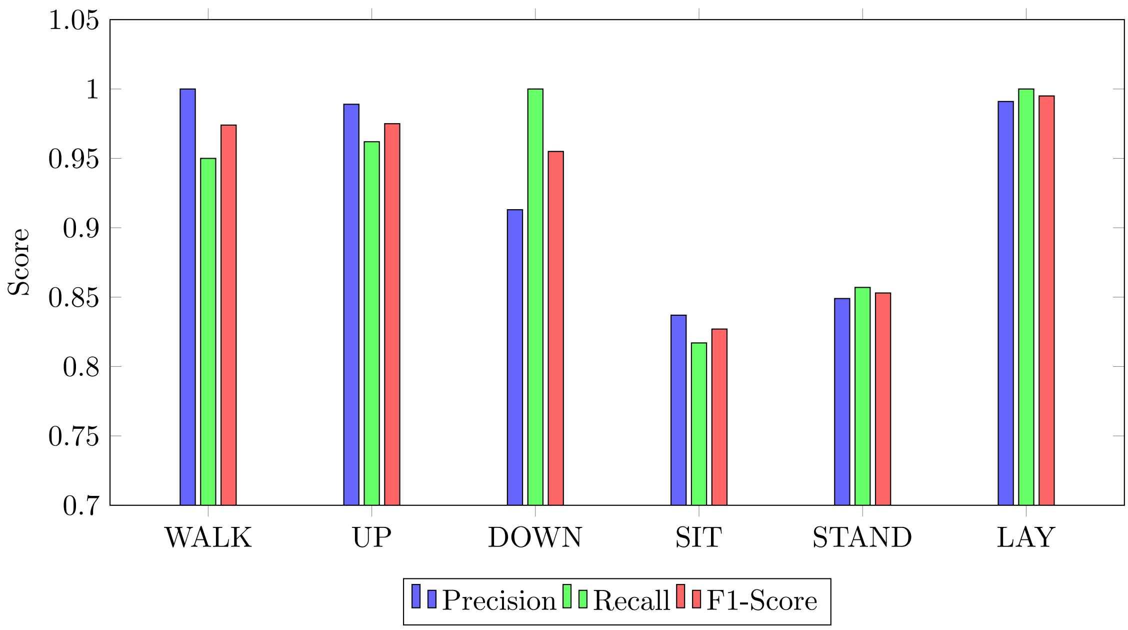
Estas dos actividades son **estáticas** (sin movimiento significativo). Los patrones de señales son muy similares:

- Ambas tienen aceleración casi constante (solo gravedad)
- Poca variación en giroscopio (sin rotación)
- La diferencia principal está en la orientación del dispositivo, que puede variar según dónde lleve el usuario el teléfono

Posibles soluciones:

1. Agregar más features (ej: ángulo promedio)
2. Usar ventanas de tiempo más largas
3. Incluir información del sensor de orientación

6.5. Gráfico de Métricas por Clase



6.6. Resumen de Observaciones

✓ **Excelente accuracy** (¿95 %) - El modelo funciona muy bien

✓ **Buena generalización** - Gap train-val ¡ 1 %

✓ **LAYING y DOWN** perfectamente clasificados (100 % recall)

✓ **WALKING** nunca se confunde con otras clases (100 % precision)

× **SITTING/STANDING** son las clases más difíciles (∼83 % F1)

7. Preguntas Potenciales del Profesor

7.1. Conceptos Fundamentales

Pregunta 1: ¿Por qué usaste convolución 1D en vez de 2D?

Respuesta: Las señales de sensores son **series temporales unidimensionales**. La convolución 1D desliza el kernel a lo largo del eje temporal, detectando patrones locales en la secuencia. La convolución 2D se usa para imágenes donde hay estructura espacial en dos dimensiones (alto y ancho). Aquí solo tenemos una dimensión temporal.

Pregunta 2: ¿Qué es un kernel y qué tamaño usaste?

Respuesta: Un kernel es un vector de pesos que se desliza sobre la señal calculando productos punto. Usé `kernel_size=5`, lo que significa que cada kernel mira 5 valores consecutivos de la señal. A 50Hz, esto equivale a 0.1 segundos de datos, suficiente para detectar patrones como un paso al caminar.

Pregunta 3: ¿Para qué sirve el padding?

Respuesta: El padding agrega ceros a los bordes de la señal para que la salida tenga el mismo tamaño que la entrada. Con `kernel_size=5` y `padding=2`:

$$\text{output\_size} = 128 - 5 + 1 + 2(2) = 128$$

Sin padding, la señal se encogerían en cada capa.

Pregunta 4: Explica qué es BatchNormalization y por qué la usaste

Respuesta: BatchNorm normaliza las activaciones de cada capa a media 0 y varianza 1 (z-score), usando las estadísticas del mini-batch actual:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Lo usé porque: (1) Estabiliza el entrenamiento, (2) Permite learning rates más altos, (3) Actúa como regularizador ligero.

Pregunta 5: ¿Qué diferencia hay entre Dropout y Weight Decay?

Respuesta:

- **Dropout**: Durante entrenamiento, apaga neuronas aleatoriamente con probabilidad p. Fuerza a la red a no depender de neuronas específicas. Es una regularización *estructural*.
- **Weight Decay (L2)**: Añade un término $\lambda||W||^2$ al loss, penalizando pesos grandes. Es una regularización *paramétrica*.

Usé ambos: Dropout(0.3) y weight_decay=10^{-2}.

7.2. Arquitectura

Pregunta 6: ¿Por qué elegiste kernel_size

Respuesta: A 50Hz de muestreo, kernel_size=5 cubre 0.1 segundos de señal. Esto es suficiente para capturar eventos locales como un paso al caminar (~0.5-1s por ciclo), transiciones de movimiento, y oscilaciones del giroscopio. Kernels más pequeños (3) capturan menos contexto; más grandes (7-11) podrían perder detalles finos.

Pregunta 7: ¿Cómo calculaste que la salida es 128×32 antes del flatten?

Respuesta: Siguiendo las dimensiones:

1. Input: (B, 9, 128)
2. Conv1 + Pool: 128 → 128/2 = 64 (MaxPool divide por 2)
3. Conv2 + Pool: 64 → 64/2 = 32
4. Canales: 128 (salida de Conv2)
5. Flatten: $128 \times 32 = 4096$

Pregunta 8: ¿Qué pasaría si quitaras MaxPooling?

Respuesta: Sin MaxPooling: (1) La dimensión temporal no se reduciría (128 en vez de 32), (2) El flatten produciría $128 \times 128 = 16384$ valores, (3) Linear1 tendría ~4.2 millones de parámetros (vs 1M actual), (4) Mayor riesgo de overfitting y más lento, (5) Se perdería la invarianza a pequeñas traslaciones temporales.

Pregunta 9: ¿Por qué usas ReLU y no otra función de activación?

Respuesta: ReLU tiene ventajas prácticas: (1) Computacionalmente eficiente (solo comparar con 0), (2) No satura como sigmoid/tanh (gradientes no se desvanecen para valores grandes), (3) Sparsity (neuronas negativas se "apagan", creando representaciones dispersas).

Pregunta 10: Explica el flujo de dimensiones desde input hasta output

Respuesta: Con batch_size=64: Input (64,9,128) → Conv1 (64,64,128) → Pool (64,64,64) → Conv2 (64,128,64) → Pool (64,128,32) → Flatten (64,4096) → FC1 (64,256) → FC2 (64,6).

7.3. Entrenamiento

Pregunta 11: ¿Qué es Cross-Entropy Loss y cómo se calcula?

Respuesta: Cross-Entropy mide la diferencia entre la distribución predicha y la real: $L = -\log(\hat{p}_{clase\_correcta})$. Ejemplo: Si predigo $\hat{p} = [0.85, 0.05, ...]$ y la clase real es 0: $L = -\log(0.85) = 0.163$. Penaliza más cuando el modelo está seguro pero equivocado.

Pregunta 12: ¿Por qué usaste AdamW en vez de SGD?

Respuesta: Adam combina Momentum (usa historia de gradientes) y RMSprop (adapta el learning rate por parámetro). AdamW mejora Adam separando el weight decay del gradiente. Converge más rápido que SGD y requiere menos tuning del learning rate.

Pregunta 13: ¿Qué es Early Stopping y por qué lo implementaste?

Respuesta: Early Stopping detiene el entrenamiento cuando el val_loss deja de mejorar por N épocas (patience=15). Lo implementé porque: (1) Previene overfitting, (2) Ahorra tiempo, (3) Restaura el mejor modelo (época 73, no la 88).

Pregunta 14: ¿Cómo elegiste el learning rate de 10^{-3}?

Respuesta: 10^{-3} es el valor por defecto recomendado para Adam/AdamW. Es un punto de partida estándar: muy alto (10^{-1}) diverge, muy bajo (10^{-5}) converge demasiado lento.

Pregunta 15: Explica qué hace la backpropagation

Respuesta: Backpropagation calcula el gradiente del loss respecto a cada peso usando la regla de la cadena: $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial W}$. Proceso: (1) Forward pass, (2) Calcular loss, (3) Propagar gradientes hacia atrás, (4) Actualizar pesos.

7.4. Resultados

Pregunta 16: ¿Por qué SITTING y STANDING se confunden tanto?

Respuesta: Ambas son actividades **estáticas**: acelerómetro solo muestra gravedad (casi constante), giroscopio sin movimiento significativo. La única diferencia es la orientación del teléfono, que varía según el bolsillo. Representan el 5.3 % de errores totales (157 de 2947).

Pregunta 17: ¿Qué significa un F1-Score de 0.95?

Respuesta: F1 es la media armónica de Precision y Recall: $F1 = 2 \cdot \frac{P \times R}{P+R}$. Un F1 de 0.95 indica alto Precision (pocas falsas alarmas), alto Recall (detectamos la mayoría), y balance entre ambos.

Pregunta 18: ¿Cómo interpretas la matriz de confusión?

Respuesta: Diagonal = predicciones correctas. Fila = distribución de predicciones para cada clase real. Columna = qué clases reales fueron predichas como esa clase. Fuera de diagonal = errores/confusiones.

Pregunta 19: ¿Qué es Precision vs Recall?

Respuesta: Precision = De mis predicciones positivas, ¿cuántas son correctas? $P = \frac{TP}{TP+FP}$. **Recall** = De los positivos reales, ¿cuántos detecté? $R = \frac{TP}{TP+FN}$.

Pregunta 20: ¿Cómo sabes que no hay overfitting?

Respuesta: Indicadores: (1) Gap train-val pequeño (97.52 % vs 97.82 %), (2) Curvas paralelas que no divergen, (3) Test accuracy (95.64 %) cercano a validation, (4) Early stopping detuvo antes de memorizar.

7.5. Matemáticas

Pregunta 21: Escribe la fórmula de la convolución 1D

Respuesta:

$$y[t] = \sum_{k=0}^{K-1} w[k] \cdot x[t+k] + b$$

Para K=5: $y[t] = w_0 x_t + w_1 x_{t+1} + w_2 x_{t+2} + w_3 x_{t+3} + w_4 x_{t+4} + b$

Pregunta 22: ¿Cómo se calcula el número de parámetros en Conv1d?

Respuesta: Params = (kernel_size × in_channels + 1) × out_channels. Para Conv1: $(5 \times 9 + 1) \times 64 = 46 \times 64 = 2,944$.

Pregunta 23: Explica la regla de la cadena en backpropagation

Respuesta: Si $L = f(g(h(W)))$, entonces: $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial W}$. Cada capa propaga su gradiente local hacia atrás multiplicándolo con el gradiente acumulado.

Pregunta 24: ¿Qué es un tensor de orden 3?

Respuesta: Un tensor de orden 3 es un arreglo tridimensional. Nuestros datos tienen shape (N, C, T): N=7,352 muestras, C=9 canales, T=128 timesteps. Acceso: $X[i, j, k]$ = valor en muestra i, canal j, tiempo k.

Pregunta 25: ¿Cómo se calcula el accuracy?

Respuesta:

$$\text{Accuracy} = \frac{\text{Predicciones correctas}}{\text{Total}} = \frac{2819}{2947} = 0.9564 = 95.64\,\%$$

8. Conclusiones

8.1. Logros del Proyecto

- ✓ Accuracy de **95.64 %** en test (objetivo cumplido)
- ✓ Buena generalización (gap train-val ¡ 1 %)
- ✓ Modelo eficiente (∼1.1M parámetros)
- ✓ Entrenamiento estable con early stopping
- ✓ 100 % recall en LAYING y WALKING_DOWNSTAIRS

8.2. Conceptos Matemáticos Aplicados

Área	**Conceptos Usados**
Álgebra Lineal	Tensores, Convolución, Multiplicación matricial
Estadística	Media/varianza, Z-score, Métricas (P, R, F1)
Cálculo	Derivadas parciales, Regla de la cadena
Teoría de Información	Cross-Entropy, Verosimilitud

8.3. Posibles Mejoras

1. **Arquitectura**: Añadir más capas conv o probar LSTM/Transformer
2. **Data augmentation**: Añadir ruido, escalar señales, time warping
3. **Features**: Añadir estadísticas (media, std, FFT) como canales extra
4. **Ensemble**: Combinar múltiples modelos
5. **Hiperparámetros**: Grid search para optimizar lr, dropout, kernel_size

Resumen Final

Se construyó exitosamente una red neuronal híbrida **CNN-1D + MLP** para clasificar actividades humanas usando señales de sensores inerciales. El modelo alcanzó un **95.64 % de accuracy** en el conjunto de test, demostrando una excelente capacidad de generalización y correcta aplicación de conceptos de álgebra lineal, estadística e inteligencia artificial.