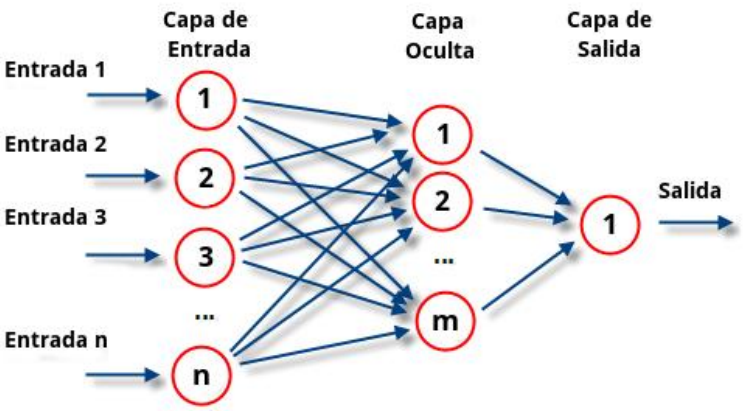


Práctica 1

USO PRÁCTICO DE RNS DENSAMENTE CONECTADAS

María Luisa Silva Mejías | Computación Inteligente | Curso 25/26

Opciones utilizadas

Entorno	Instalación local
Dataset	Fashion-MNIST precargado de Keras
Activación	ReLU
Optimizador	Adam
Número de configuraciones	3

Tabla de configuraciones:

Modelo	**Capas ocultas**	**Neuronas**	***Epochs***	***Batch size***	**LR**
1	2	256-128	10	64	0,001
2	2	128-64	15	32	0,0008
3	3	512-256-128	20	128	0,0005

Proceso de desarrollo

Imports

```python
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras import models, layers
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.metrics import Precision, Recall
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K

import numpy as np
import seaborn as sns
```

✓ 14.5s Python

En este apartado se importan las librerías necesarias para el desarrollo del experimento.
Entre ellas se incluyen Keras para crear y entrenar las redes neuronales, *NumPy* para operaciones numéricas, *Seaborn* y *Matplotlib* para la visualización de matrices de confusión, y *Scikit-learn* para generar las métricas basadas en predicciones.

1. Cargar el dataset Fashion-MNIST

```python
# Cargar datos
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalizar (0-255 → 0-1)
x_train = x_train / 255.0
x_test = x_test / 255.0

# Aplanar imágenes: 28x28 → 784
x_train = x_train.reshape(-1, 28*28)
x_test = x_test.reshape(-1, 28*28)

# Convertir etiquetas a one-hot
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

✓ 0.5s Python

Se carga el conjunto de datos Fashion-MNIST, que viene preincorporado en Keras.

Las imágenes se normalizan al rango [0, 1] dividiendo cada píxel por 255, y posteriormente se aplanan de 28×28 a un vector de 784 valores para poder ser usadas en capas densas.

Las etiquetas originales (de 0 a 9) se convierten al formato *one-hot encoding* porque se utiliza la función de pérdida *categorical crossentropy*, que requiere este formato.

2. Modelo base y variantes

Modelo base (256 y 128 neuronas)

Definir el modelo

```
model1 = models.Sequential([
    layers.Dense(256, activation='relu', input_shape=(784,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model1.summary()
```

✓ 0.1s Python

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               200960

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
```

Compilar el modelo y configurar el aprendizaje

```
model1.compile(optimizer=Adam(learning_rate=0.001),
               loss='categorical_crossentropy',
               metrics=['accuracy', Precision(), Recall()])
```

✓ 0.0s Python

Entrenar o ajustar (fit) el modelo

```
history1 = model1.fit(x_train, y_train, epochs=10, batch_size=64)
```

✓ 13.8s Python

```
Epoch 1/10
938/938 [==============================] - 2s 1ms/step - loss: 0.4904 - accuracy: 0.8251 - precision: 0.8729 - recall: 0.7739
Epoch 2/10
938/938 [==============================] - 1s 1ms/step - loss: 0.3619 - accuracy: 0.8673 - precision: 0.8952 - recall: 0.8407
Epoch 3/10
938/938 [==============================] - 1s 1ms/step - loss: 0.3253 - accuracy: 0.8810 - precision: 0.9048 - recall: 0.8580
Epoch 4/10
938/938 [==============================] - 1s 1ms/step - loss: 0.3006 - accuracy: 0.8882 - precision: 0.9089 - recall: 0.8688
Epoch 5/10
938/938 [==============================] - 1s 1ms/step - loss: 0.2829 - accuracy: 0.8947 - precision: 0.9140 - recall: 0.8766
Epoch 6/10
938/938 [==============================] - 1s 1ms/step - loss: 0.2703 - accuracy: 0.8980 - precision: 0.9165 - recall: 0.8820
Epoch 7/10
938/938 [==============================] - 1s 1ms/step - loss: 0.2571 - accuracy: 0.9034 - precision: 0.9201 - recall: 0.8881
Epoch 8/10
938/938 [==============================] - 1s 2ms/step - loss: 0.2456 - accuracy: 0.9075 - precision: 0.9224 - recall: 0.8938
Epoch 9/10
938/938 [==============================] - 1s 1ms/step - loss: 0.2342 - accuracy: 0.9123 - precision: 0.9267 - recall: 0.8996
Epoch 10/10
938/938 [==============================] - 1s 2ms/step - loss: 0.2287 - accuracy: 0.9144 - precision: 0.9278 - recall: 0.9023
```

Este modelo se compone de dos capas densas ocultas con 256 y 128 neuronas respectivamente, ambas con activación ReLU.

Como capa de salida se utiliza una softmax de 10 neuronas (una por clase).

Se entrena durante 10 épocas, con *batch size* de 64 y el optimizador Adam con *learning rate* = 0,001.

Como se observa en la salida de entrenamiento, la función de pérdida disminuye de manera constante a lo largo de las épocas, mientras que las demás métricas aumentan progresivamente. Esto indica que el modelo aprende de forma estable y mejora su rendimiento en cada iteración.

Modelo 2 (128 y 64 neuronas)

Definir el modelo

```
model2 = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(784,)),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model2.summary()
```

✓ 0.0s Python

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               100480

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
```

Compilar el modelo y configurar el aprendizaje

```
model2.compile(optimizer=Adam(learning_rate=0.0008),
               loss='categorical_crossentropy',
               metrics=['accuracy', Precision(), Recall()])
```

✓ 0.0s Python

Entrenar o ajustar (fit) el modelo

```
history2 = model2.fit(x_train, y_train, epochs=15, batch_size=32)
```

✓ 30.6s Python

```
Epoch 1/15
1875/1875 [==============================] - 2s 1ms/step - loss: 0.5019 - accuracy: 0.8234 - precision_1: 0.8727 - recall_1: 0.7680
Epoch 2/15
1875/1875 [==============================] - 2s 944us/step - loss: 0.3720 - accuracy: 0.8656 - precision_1: 0.8945 - recall_1: 0.8367
Epoch 3/15
1875/1875 [==============================] - 2s 991us/step - loss: 0.3343 - accuracy: 0.8773 - precision_1: 0.9030 - recall_1: 0.8539
Epoch 4/15
1875/1875 [==============================] - 2s 942us/step - loss: 0.3129 - accuracy: 0.8854 - precision_1: 0.9085 - recall_1: 0.8634
Epoch 5/15
1875/1875 [==============================] - 2s 995us/step - loss: 0.2947 - accuracy: 0.8920 - precision_1: 0.9121 - recall_1: 0.8726
Epoch 6/15
1875/1875 [==============================] - 2s 933us/step - loss: 0.2806 - accuracy: 0.8964 - precision_1: 0.9152 - recall_1: 0.8787
Epoch 7/15
1875/1875 [==============================] - 2s 995us/step - loss: 0.2666 - accuracy: 0.9013 - precision_1: 0.9192 - recall_1: 0.8845
Epoch 8/15
1875/1875 [==============================] - 2s 981us/step - loss: 0.2554 - accuracy: 0.9051 - precision_1: 0.9219 - recall_1: 0.8900
Epoch 9/15
1875/1875 [==============================] - 2s 994us/step - loss: 0.2455 - accuracy: 0.9079 - precision_1: 0.9231 - recall_1: 0.8937
Epoch 10/15
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2389 - accuracy: 0.9104 - precision_1: 0.9249 - recall_1: 0.8966
Epoch 11/15
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2307 - accuracy: 0.9132 - precision_1: 0.9262 - recall_1: 0.8999
Epoch 12/15
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2210 - accuracy: 0.9160 - precision_1: 0.9288 - recall_1: 0.9046
Epoch 13/15
...
Epoch 14/15
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2073 - accuracy: 0.9210 - precision_1: 0.9325 - recall_1: 0.9098
Epoch 15/15
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2026 - accuracy: 0.9234 - precision_1: 0.9349 - recall_1: 0.9133
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Este modelo también tiene dos capas ocultas, pero con menor complejidad: 128 y 64 neuronas.

Se entrena durante 15 épocas, con *batch size* de 32 y un *learning rate* ligeramente reducido (0,0008) para estabilizar el aprendizaje en configuraciones más pequeñas.

En este modelo también se observa una disminución progresiva de la función de pérdida y un incremento continuo del *accuracy*, *precision* y *recall*, lo que indica un entrenamiento estable.

Modelo 3 (512, 256 y 128 neuronas)

Definir el modelo

```
model3 = models.Sequential([
    layers.Dense(512, activation='relu', input_shape=(784,)),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

✓ 0.0s Python

Compilar el modelo y configurar el aprendizaje

```
model3.compile(optimizer=Adam(learning_rate=0.0005),
               loss='categorical_crossentropy',
               metrics=['accuracy', Precision(), Recall()])
```

✓ 0.0s Python

Entrenar o ajustar (fit) el modelo

```
history3 = model3.fit(x_train, y_train, epochs=20, batch_size=128)
```

✓ 45.0s Python

```
Epoch 1/20
469/469 [==============================] - 2s 4ms/step - loss: 0.5140 - accuracy: 0.8183 - precision_2: 0.8759 - recall_2: 0.7564
Epoch 2/20
469/469 [==============================] - 2s 4ms/step - loss: 0.3608 - accuracy: 0.8695 - precision_2: 0.8981 - recall_2: 0.8402
Epoch 3/20
469/469 [==============================] - 2s 5ms/step - loss: 0.3189 - accuracy: 0.8826 - precision_2: 0.9063 - recall_2: 0.8596
Epoch 4/20
469/469 [==============================] - 2s 4ms/step - loss: 0.2947 - accuracy: 0.8918 - precision_2: 0.9134 - recall_2: 0.8718
Epoch 5/20
469/469 [==============================] - 2s 4ms/step - loss: 0.2749 - accuracy: 0.8978 - precision_2: 0.9158 - recall_2: 0.8803
Epoch 6/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2596 - accuracy: 0.9015 - precision_2: 0.9197 - recall_2: 0.8859
Epoch 7/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2484 - accuracy: 0.9067 - precision_2: 0.9233 - recall_2: 0.8921
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.2338 - accuracy: 0.9112 - precision_2: 0.9258 - recall_2: 0.8982
Epoch 9/20
469/469 [==============================] - 3s 6ms/step - loss: 0.2250 - accuracy: 0.9155 - precision_2: 0.9284 - recall_2: 0.9028
Epoch 10/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2147 - accuracy: 0.9186 - precision_2: 0.9317 - recall_2: 0.9074
Epoch 11/20
469/469 [==============================] - 2s 5ms/step - loss: 0.2040 - accuracy: 0.9219 - precision_2: 0.9335 - recall_2: 0.9112
Epoch 12/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1969 - accuracy: 0.9250 - precision_2: 0.9361 - recall_2: 0.9156
Epoch 13/20
...
Epoch 19/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1434 - accuracy: 0.9451 - precision_2: 0.9510 - recall_2: 0.9398
Epoch 20/20
469/469 [==============================] - 2s 5ms/step - loss: 0.1401 - accuracy: 0.9476 - precision_2: 0.9538 - recall_2: 0.9422
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Incluye tres capas densas ocultas con 512, 256 y 128 neuronas respectivamente, siendo el modelo más profundo y complejo del experimento.

Se entrena durante 20 épocas, con *batch size* de 128 y un *learning rate* más pequeño (0,0005) para evitar inestabilidad debido a su mayor tamaño.

En este modelo también se observa una disminución constante de la función de pérdida y un aumento de las métricas, aunque en este caso la reducción del *loss* es mayor y se alcanzan valores de precisión y *recall* superiores.

3. Evaluación

Función F1-Score

```python
def f1_score(recall, precision):
    return 2 * ((precision * recall) / (precision + recall + K.epsilon()))
```

0.0s Python

```python
test_loss1, test_acc1, test_prec1, test_rec1 = model1.evaluate(x_test, y_test)
test_f1score1 = f1_score(test_rec1, test_prec1)

test_loss2, test_acc2, test_prec2, test_rec2 = model2.evaluate(x_test, y_test)
test_f1score2 = f1_score(test_rec2, test_prec2)

test_loss3, test_acc3, test_prec3, test_rec3 = model3.evaluate(x_test, y_test)
test_f1score3 = f1_score(test_rec3, test_prec3)

print("F1-Scores:")
print(f"Modelo 1 - {test_f1score1}")
print(f"Modelo 2 - {test_f1score2}")
print(f"Modelo 3 - {test_f1score3}")

predictions1 = model1.predict(x_test)
predictions2 = model2.predict(x_test)
predictions3 = model3.predict(x_test)

# Obtener las clases predichas
predicted_classes1 = np.argmax(predictions1, axis=1)
predicted_classes2 = np.argmax(predictions2, axis=1)
predicted_classes3 = np.argmax(predictions3, axis=1)

true_classes = np.argmax(y_test, axis=1)
class_labels = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Graficar la matriz de confusión 1
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.4)  # Aumentar el tamaño de la fuente para la matriz de confusión
sns.heatmap(confusion_matrix(true_classes, predicted_classes1), annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title('Matriz de Confución Modelo 1')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()

# Graficar la matriz de confusión 2
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.4)  # Aumentar el tamaño de la fuente para la matriz de confusión
sns.heatmap(confusion_matrix(true_classes, predicted_classes2), annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title('Matriz de Confución Modelo 2')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()

# Graficar la matriz de confusión 3
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.4)  # Aumentar el tamaño de la fuente para la matriz de confusión
sns.heatmap(confusion_matrix(true_classes, predicted_classes3), annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.title('Matriz de Confución Modelo 3')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```

3.5s Python

```
313/313 [==============================] - 0s 824us/step - loss: 0.3135 - accuracy: 0.8898 - precision: 0.9026 - recall: 0.8784
313/313 [==============================] - 0s 706us/step - loss: 0.3328 - accuracy: 0.8850 - precision_1: 0.9021 - recall_1: 0.8737
313/313 [==============================] - 1s 1ms/step - loss: 0.3580 - accuracy: 0.8934 - precision_2: 0.9045 - recall_2: 0.8872
F1-Scores:
Modelo 1 - 0.8903303846662579
Modelo 2 - 0.8876809279068681
Modelo 3 - 0.8957544179182184
313/313 [==============================] - 0s 696us/step
313/313 [==============================] - 0s 599us/step
313/313 [==============================] - 1s 1ms/step
```

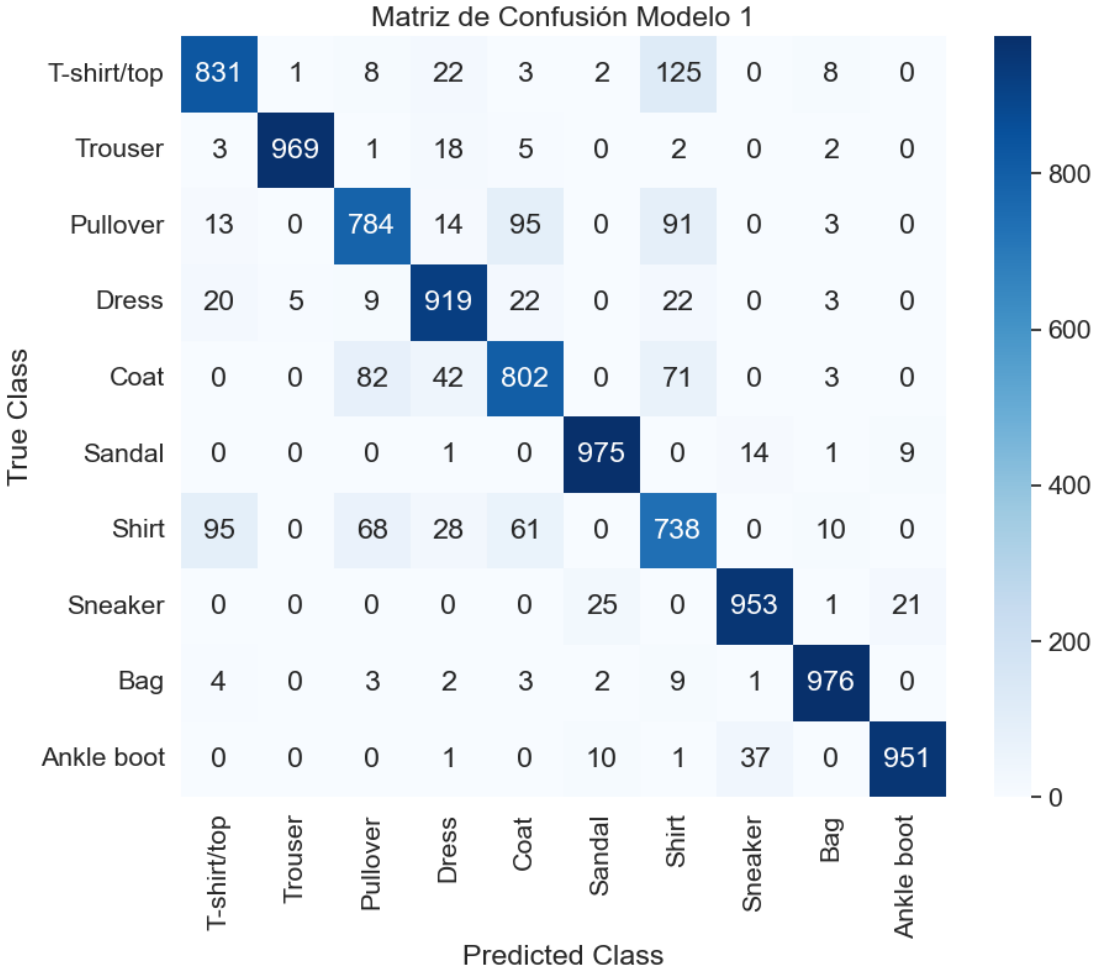
Tras el entrenamiento, cada modelo se evalúa con el conjunto de test usando:

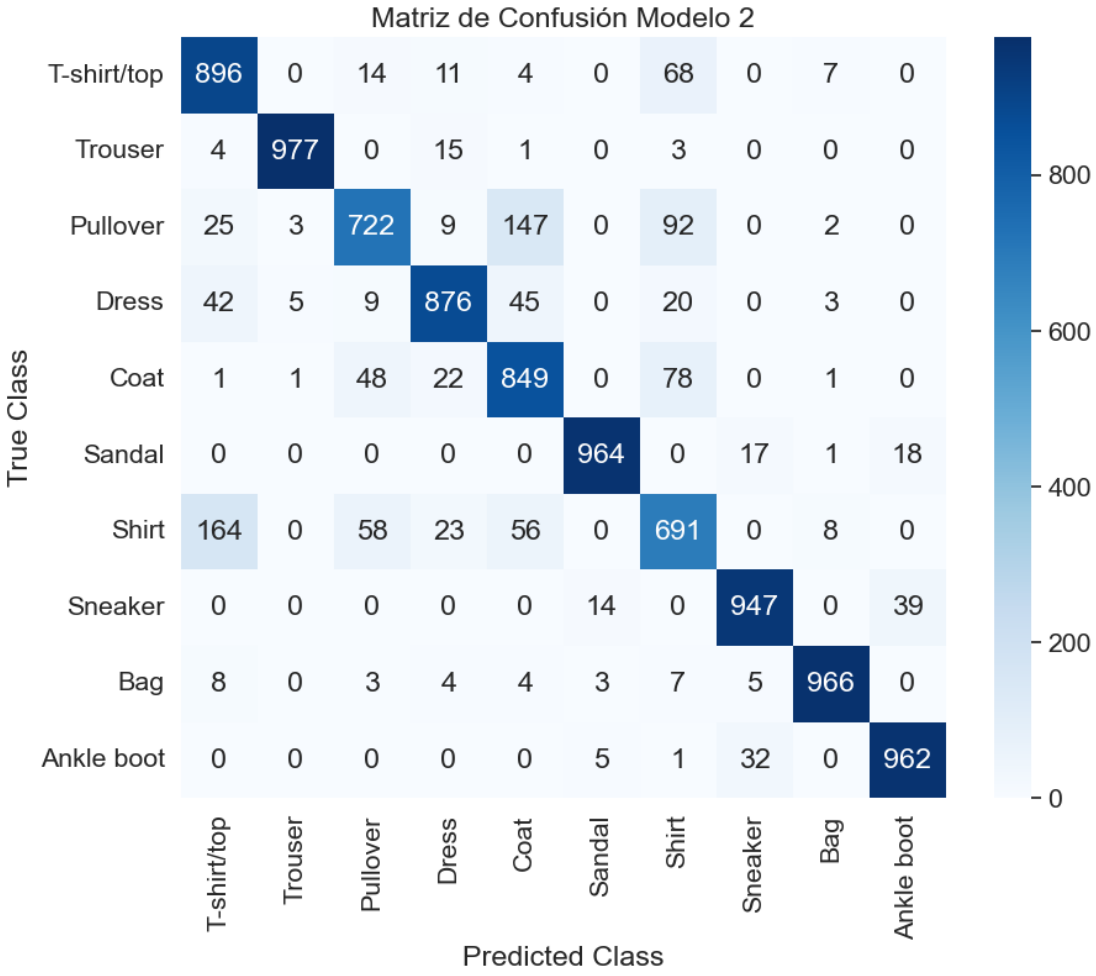
- *Accuracy*
- *Precision*

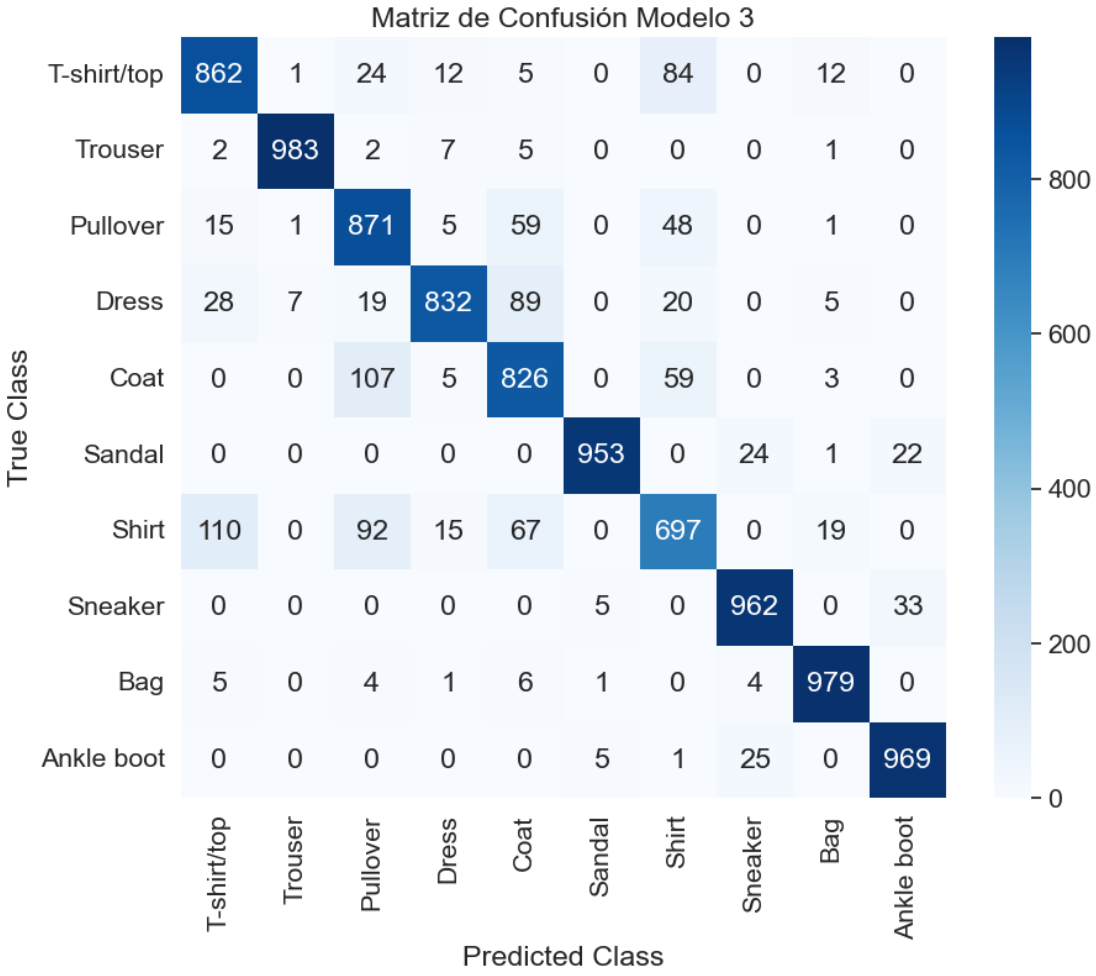
- *Recall*
- *F1-Score*, que se calcula manualmente mediante la fórmula estándar:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Esto permite comparar los tres modelos no solo en términos de acierto global, sino también considerando el equilibrio entre falsos positivos y falsos negativos.




Matriz de Confusión Modelo 2
T-shirt/top 896 0 14 11 4 0 68 0 7 0
Trouser 4 977 0 15 1 0 3 0 0 0
Pullover 25 3 722 9 147 0 92 0 2 0
Dress 42 5 9 876 45 0 20 0 3 0
Coat 1 1 48 22 849 0 78 0 1 0
Sandal 0 0 0 0 0 964 0 17 1 18
Shirt 164 0 58 23 56 0 691 0 8 0
Sneaker 0 0 0 0 0 14 0 947 0 39
Bag 8 0 3 4 4 3 7 5 966 0
Ankle boot 0 0 0 0 0 5 1 32 0 962
True Class
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
Predicted Class
800
600
400
200
0



Para cada modelo se genera una matriz de confusión, representada como un mapa de calor mediante *Seaborn*.

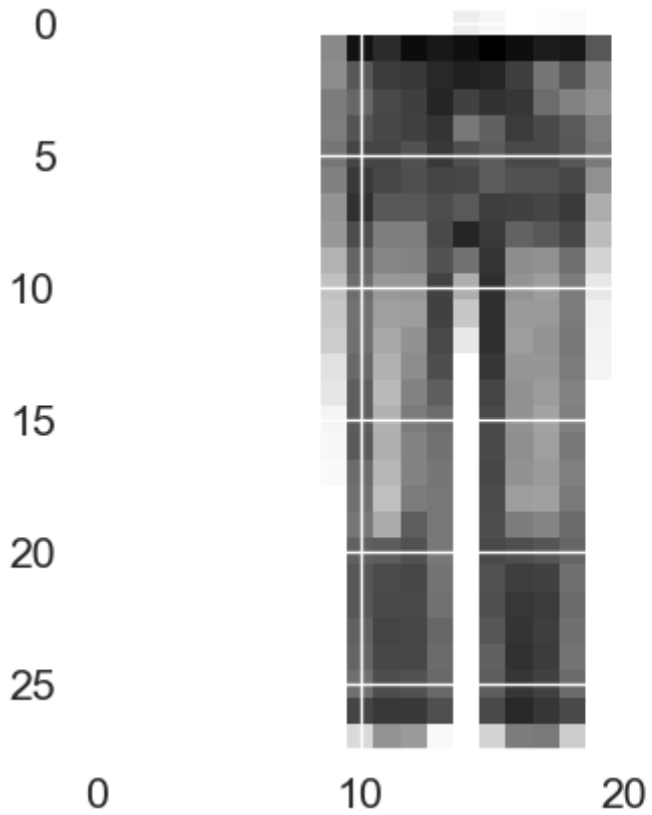
Estas matrices permiten analizar qué clases se confunden más entre sí, proporcionando una evaluación más detallada que la *accuracy*.

4. Usar el modelo

```
plt.imshow(x_test[15].reshape(28, 28), cmap=plt.cm.binary)

print("Valor más alto (mayor probabilidad) para el elemento 11:")
print(f"Modelo 1 predice la clase: {class_labels[np.argmax(predictions1[15])]}")
print(f"Modelo 2 predice la clase: {class_labels[np.argmax(predictions2[15])]}")
print(f"Modelo 3 predice la clase: {class_labels[np.argmax(predictions3[15])]}\n")

print("Etiqueta real:")
print(f"{class_labels[np.argmax(y_test[15])]}\n")

print(f"Array de probabilidades del modelo 1: {predictions1[15]}")
print(f"Array de probabilidades del modelo 2: {predictions2[15]}")
print(f"Array de probabilidades del modelo 3: {predictions3[15]}")
```

✓ 0.0s Python

```
Valor más alto (mayor probabilidad) para el elemento 11:
Modelo 1 predice la clase: Trouser
Modelo 2 predice la clase: Trouser
Modelo 3 predice la clase: Trouser

Etiqueta real:
Trouser

Array de probabilidades del modelo 1: [2.2585251e-08 9.9999976e-01 4.0623012e-09 1.5961160e-07 7.6444280e-08
 3.6201741e-14 1.7369608e-08 4.5202490e-16 1.5596847e-09 4.3574949e-14]
Array de probabilidades del modelo 2: [8.9221217e-09 9.9999881e-01 1.8398313e-10 3.9898239e-07 7.2018167e-08
 5.7282022e-12 1.9170863e-09 2.3691745e-12 7.0435294e-07 2.2305302e-11]
Array de probabilidades del modelo 3: [1.2134601e-11 1.0000000e+00 9.4589338e-15 1.0311534e-11 1.1043018e-10
 7.8833356e-14 2.4762140e-11 4.2743678e-16 1.9776515e-14 9.5317924e-19]
```



Finalmente, se visualiza una imagen del conjunto de test (en este caso se ha cogido la número 15 como ejemplo) y se muestra qué clase predice cada modelo junto con las probabilidades completas de salida.
Este apartado ilustra cómo funciona la red en la práctica y permite observar diferencias entre los tres modelos a nivel de confianza.

Los tres modelos predicen correctamente la clase del ejemplo analizado (*Trouser*). Además, los vectores de probabilidad muestran que, a medida que aumenta la complejidad del modelo, la confianza de la predicción también se incrementa, siendo el modelo 3 el que ofrece la probabilidad más cercana al 100%.

Tabla de resultados

La siguiente tabla resume los resultados obtenidos por cada una de las configuraciones evaluadas. Esto permite comparar fácilmente el efecto de los cambios en la arquitectura y en los hiperparámetros sobre el rendimiento del modelo.

Modelo	*Loss*	*Accuracy*	Precisión	Sensibilidad (*recall*)	*F1-score*
1 (256-128)	0,3135	0,8898	0,9026	0,8784	0,8903
2 (128-64)	0,3328	0,8850	0,9021	0,8737	0,8877
3 (512-256-128)	0,3580	0,8934	0,9045	0,8872	0,8958

Análisis de la tabla de resultados

Los tres modelos evaluados muestran un rendimiento competitivo, pero con diferencias relevantes derivadas de la arquitectura, del número de neuronas por capa y de los hiperparámetros.

El modelo 1 presenta un equilibrio sólido entre complejidad y rendimiento. Obtiene un *loss* relativamente bajo (0,3135) y métricas bastante homogéneas: *accuracy* del 88,98%, precisión del 90,26% y un *F1-score* del 89,03%. Estos resultados indican que la arquitectura base ofrece una buena capacidad de generalización.

En el modelo 2, al reducir el número de neuronas, el rendimiento disminuye ligeramente. Su *loss* es mayor (0,3328) y también baja la *accuracy* (88,50%) junto con el F1-score (88,77%). Aunque sigue siendo un modelo estable, esta configuración evidencia que una arquitectura demasiado pequeña pierde capacidad para capturar patrones complejos en los datos, lo que se refleja en una menor sensibilidad (87,37%).

A pesar de que su *loss* no es el más bajo (0,3580), el modelo 3 logra las mejores métricas globales: *accuracy* (89,34%), precisión (90,45%), sensibilidad (88,72%) y el *F1-score* más alto (89,58%). Esto indica que, aunque es un modelo más profundo y con mayor capacidad, es capaz de aprender mejor las características de las imágenes y ofrecer un rendimiento ligeramente superior en términos de clasificación. Su *loss* algo mayor puede deberse a una convergencia diferente, pero no afecta negativamente al resto de métricas.

En conclusión, el modelo 2 es el que presenta el peor desempeño, algo esperable debido a su menor capacidad. En comparación, el modelo 1 consigue un equilibrio mucho más sólido entre simplicidad y rendimiento, logrando mejores métricas sin

necesidad de aumentar demasiado la complejidad. Por su parte, el modelo 3 destaca claramente sobre ambos, ya que obtiene los mejores resultados en *accuracy*, precisión, *recall* y *F1-score*. Esto lo convierte en la opción más prometedora siempre que se disponga de suficiente capacidad computacional para entrenarlo y ejecutarlo.