SENATI

EXAMEN FINAL DE MACHINE LEARNING Y DEEP LEARNING

Ejercicio 1:

Carga el dataset MNIST desde tensorflow.keras.datasets.

- Muestra las dimensiones de los conjuntos de entrenamiento y prueba.
- Visualiza 10 imágenes con sus etiquetas reales.

Puedes descargarlo desde el sitio oficial de Yann LeCun:
http://yann.lecun.com/exdb/mnist/

Allí encontrarás archivos como:

- train-images-idx3-ubyte.gz
- train-labels-idx1-ubyte.gz
- t10k-images-idx3-ubyte.gz
- t10k-labels-idx1-ubyte.gz

Ejercicio 2:
Normaliza los valores de los píxeles (0–255 → 0–1).
Convierte las etiquetas en formato *one-hot encoding* utilizando to_categorical.

Objetivo: Aplicar técnicas básicas de preprocesamiento para redes neuronales.

Ejercicio 3:
Entrena un **clasificador SVM (Support Vector Machine)** con un subconjunto de 10,000 imágenes.
Evalúa su exactitud en el conjunto de prueba.

Objetivo: Contrastar un modelo clásico de ML frente a redes neuronales.

Ejercicio4:
Construye una **Red Neuronal Multicapa (MLP)** con:

- Capa de entrada de 784 neuronas (28x28)
- 2 capas ocultas (128 y 64 neuronas, activación ReLU)
- Capa de salida (10 neuronas, activación softmax)

Entrena durante 10 épocas y evalúa la precisión.

Objetivo: Implementar una red totalmente conectada.

Ejercicio5:
Modifica el número de neuronas y el optimizador (por ejemplo: adam, sgd, rmsprop).
Evalúa cómo cambia la exactitud en validación.

Objetivo: Entender el impacto de los hiperparámetros en el rendimiento.

Ejercicio6:
Implementa una **CNN** con la siguiente arquitectura:

- Conv2D(32, kernel=3x3) + ReLU + MaxPooling(2x2)
- Conv2D(64, kernel=3x3) + ReLU + MaxPooling(2x2)
- Flatten + Dense(128, ReLU)
- Output(10, softmax)

Entrena y reporta precisión, pérdida y matriz de confusión.

Objetivo: Implementar una CNN básica para reconocimiento de imágenes.

Ejercicio 7:
Agrega Dropout(0.5) y BatchNormalization() en tu CNN.
Compara los resultados antes y después.

Objetivo: Aplicar técnicas para reducir *overfitting*.

Ejercicio 8:
Usa ImageDataGenerator para realizar *data augmentation* (rotación, desplazamiento, zoom).
Entrena la CNN y observa si mejora la precisión.

Objetivo: Incrementar la capacidad generalizadora del modelo.

Ejercicio 9:
Genera un **informe de clasificación (classification report)** y **matriz de confusión**.
Identifica los dígitos con más errores y explica brevemente por qué podría ocurrir.

Objetivo: Evaluar el modelo de forma más profunda.

Ejercicio 10:
Guarda el modelo entrenado en formato .h5.
Cárgalo nuevamente y prueba su desempeño en un conjunto de imágenes nuevas.

Objetivo: Aprender a guardar y reutilizar modelos en producción.

Para la entrega:

- Un archivo examen_MNIST.ipynb con:
  - Código ejecutable y bien comentado
  - Gráficos y resultados
  - Conclusión final: ¿qué modelo funcionó mejor y por qué?

Ejercicio 11:

Objetivo: Crear una API para servir el modelo MNIST.

Tareas:

1. Instalar Flask (!pip install flask si usas Colab local o entorno externo).
2. Cargar el modelo guardado (modelo_mnist.h5).
3. Definir una ruta /predict que reciba una imagen en formato base64 o archivo .png.
4. Preprocesar la imagen a formato 28x28 y devolver la predicción en JSON.

Ejemplo de código base:

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np
import cv2
import io
from PIL import Image

app = Flask(__name__)
model = load_model("modelo_mnist.h5")

@app.route('/predict', methods=['POST'])
def predict():
    file = request.files['file']
    image = Image.open(file).convert('L')
    image = image.resize((28, 28))
    image = np.array(image).astype('float32') / 255.0
    image = image.reshape(1, 28, 28, 1)

    prediction = np.argmax(model.predict(image))
    return jsonify({'prediccion': int(prediction)})

if __name__ == '__main__':
    app.run(debug=True)
```

Nota: Para probar la API, puedes usar curl, Postman o un formulario HTML simple que suba imágenes.