```
# Importar las librerías necesarias

!pip install tensorflow
!pip install tensorflowjs
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import tensorflowjs as tfjs
import ipywidgets as widgets
from IPython.display import display
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.utils import plot_model
import matplotlib.image as mpimg
import networkx as nx

# Definir las características (features) y los objetivos (targets)
features = np.array([
    (12.53,32.86,67.9,19.9,13.86,85.84,4.67,1.31),
(11.68,29.75,66.37,23.45,1.83,184.03,5.32,1.39),
(11.23,30.17,67.34,18.51,1.30,97.41,6.46,1.31),
(13.21,31.07,64.96,22.99,1.12,34.18,5.72,1.37),
(11.78,30.63,67.25,20.78,1.32,73.59,4.02,1.27),
(16.75,40.42,66.57,21.45,4.12,685.59,15.62,1.02),
(14.42,34.19,63.47,19.58,1.61,92.03,9.71,1.26),
(12.82,31.46,67.73,19.19,1.21,53.17,4.34,1.37),
(13.00,32.46,64.25,19.69,2.55,50.58,14.99,1.10),
(12.79,31.91,66.59,22.85,1.17,85.76,4.96,1.24),
(13.58,34.31,67.35,19.17,1.12,39.68,4.85,1.34),
(11.79,31.00,66.07,22.69,1.36,150.78,3.69,1.24),
(14.59,34.38,63.1,23.09,2.22,125.94,13.16,1.21),
(13.16,32.32,67.31,20.35,1.21,30.75,6.18,1.40),
(14.75,39.21,68.09,24.38,2.28,60.95,9.41,1.60),
(13.25,31.40,64.79,18.62,1.90,137.70,11.49,1.40),
(13.51,34.88,67.03,23.32,1.47,40.81,5.65,1.22),
(10.49,27.68,67.01,17.31,1.09,57.82,4.26,1.35),
(14.43,32.07,61.14,23.04,1.35,74.70,8.91,1.35),
(13.30,34.15,66.7,22.19,1.15,64.45,5.58,1.20),
(12.99,30.46,64.84,21.88,1.17,58.61,2.69,1.41),
(17.06,43.00,66.86,21.90,4.06,1487.56,11.56,1.14),
(11.25,30.22,67.09,19.54,2.67,247.61,6.94,1.15),
(11.67,39.87,66.71,18.26,1.83,25.09,6.53,1.01),
(11.78,33.46,67.66,20.69,1.35,203.54,4.56,1.27),
(13.66,31.40,63.39,20.52,1.26,43.88,7.94,1.32),
(12.51,33.43,69.65,22.52,2.04,161.39,7.87,1.14),
(13.05,31.98,65.49,20.27,1.66,40.91,3.69,1.20),
(13.19,29.68,60.79,19.20,1.74,91.20,11.79,1.42),
(12.55,32.48,68.52,22.21,1.82,92.16,7.04,1.15),
(12.49,32.29,65.89,20.94,2.07,317.70,10.22,1.20),
(13.86,35.61,66.47,20.79,1.52,63.76,6.65,1.22),
```

```
(14.33,37.84,66.79,19.85,1.38,38.66,3.62,1.21),
(13.87,34.37,65.73,21.11,2.17,410.54,13.42,1.29),
(14.25,33.29,61.63,21.47,1.55,156.01,7.53,1.23),
(14.22,34.75,64.6,22.8,1.68,324.27,8.04,1.17)
])

targets = np.array([
    [-2.73,0.16],
    [26.14,0.13],
    [26.81,0.0],
    [-11.92,0.21],
    [-0.23,0.10],
    [2.80,0.24],
    [-25.55,0.23],
    [-4.22,0.13],
    [-52.57,0.10],
    [-13.41,0.16],
    [-5.33,0.18],
    [-17.08,-0.02],
    [-2.93,0.22],
    [-31.76,0.18],
    [51.87,0.21],
    [-26.54,0.00],
    [46.11,0.19],
    [-31.62,0.06],
    [-49.11,0.19],
    [-0.94,0.23],
    [-13.74,0.26],
    [29.97,0.22],
    [22.66,0.15],
    [1.73,0.05],
    [55.07,0.17],
    [-40.76,0.22],
    [-4.41,0.08],
    [-3.44,0.21],
    [-33.04,0.18],
    [14.32,0.05],
    [-21.32,0.13],
    [24.64,0.18],
    [16.82,0.22],
    [4.39,0.17],
    [-33.37,0.23],
    [11.63,0.24]
])

# Normalización de los datos de entrada
scaler = MinMaxScaler()
features_normalized = scaler.fit_transform(features)
```

```
# Definir el modelo de la red neuronal con capas ocultas
adicionales
modelo = tf.keras.Sequential([
   tf.keras.layers.Dense(units=16, input_shape=[8],
activation='relu'),  # Primera capa oculta
    tf.keras.layers.Dropout(0.5),  # Aplicar Dropout con tasa 0.3
(30%)
    tf.keras.layers.Dense(units=16, activation='relu'),  # Segunda
capa oculta
    tf.keras.layers.Dropout(0.5),  # Dropout en la segunda capa
    tf.keras.layers.Dense(units=16, activation='relu'),  # Tercera
capa oculta
    tf.keras.layers.Dropout(0.5),  # Dropout en la tercera capa
    tf.keras.layers.Dense(units=2)  # Capa de salida sin Dropout
])



# Compilar el modelo
modelo.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss='mean_squared_error'
)


# Configurar el Early Stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',     # Monitorea la pérdida en el conjunto
de validación
    patience=10,            # Número de épocas sin mejora para
detener el entrenamiento
    restore_best_weights=True  # Restaura los pesos de la mejor
época
)

# Entrenar el modelo con Early Stopping
print('Inicio de entrenamiento...')
historial = modelo.fit(
    features_normalized, targets,
    epochs=1000,
    verbose=False,
    validation_split=0.2,
    callbacks=[early_stopping]  # Añadir el callback Early Stopping
)
print('Modelo entrenado!')
```

```
# Graficar la magnitud de la pérdida durante el entrenamiento
plt.xlabel('#Época')
plt.ylabel('Magnitud de pérdida')
plt.plot(historial.history['loss'], label='Entrenamiento')
plt.plot(historial.history['val_loss'], label='Validación')
plt.legend()
plt.show()


# Mostrar los pesos y sesgos de cada capa
for i, layer in enumerate(modelo.layers):
    # Check if the layer has weights (Dense layers have weights,
Dropout layers do not)
    if hasattr(layer, 'get_weights') and len(layer.get_weights()) >
0:
        pesos, sesgos = layer.get_weights()
        print(f"Capa {i+1}:")
        print(f"Pesos:\n{pesos}")
        print(f"Sesgos:\n{sesgos}")
    else:
        print(f"Capa {i+1} (Dropout) no tiene pesos ni sesgos.")
```

```
# Guardar el modelo completo en un archivo .h5
modelo.save('/content/model.h5')

# Convertir y guardar el modelo en formato TensorFlow.js
tfjs.converters.save_keras_model(modelo, '/content/MODELOS/Urbana')

# Función para hacer predicciones con el modelo entrenado
def make_prediction(RentanetaPersonamE, RentaHogarmE, activa,
edupoblacion64, Preciomediovivienda2023mE, mViviendas, Compacidad,
Numviv_Numhogares):
    # Normalizar los datos de entrada antes de predecir
    input_data = np.array([[float(RentanetaPersonamE),
float(RentaHogarmE), float(activa), float(edupoblacion64),
float(Preciomediovivienda2023mE), float(mViviendas),
float(Compacidad), float(Numviv_Numhogares)]])
    input_data_normalized = scaler.transform(input_data)
```

```
    prediction = modelo.predict(input_data_normalized)
    return prediction[0][0], prediction[0][1]

    # Mostrar los cálculos internos de la red
    print("Predicción final (sin activar):")
    for i, layer in enumerate(modelo.layers[:-1]):
        output_layer = layer(input_data_normalized).numpy()
        print(f"Salida de la capa {i+1} (después de
activación):\n{output_layer}")
        input_data_normalized = output_layer

    final_output = modelo.layers[-1](input_data_normalized).numpy()
    print(f"Salida de la capa de salida (final):\n{final_output}")

    return prediction[0][0], prediction[0][1]

# Calcular el error de predicción (MSE)
def calcular_error():
    predicciones = modelo.predict(features_normalized)
    mse = np.mean(np.square(predicciones - targets))
    print(f"Error cuadrático medio (MSE): {mse}")

# Función para graficar los valores reales contra las predicciones
def graficar_predicciones():
    # Realizar predicciones sobre los datos normalizados
    predicciones = modelo.predict(features_normalized)

    # Graficar
    plt.figure(figsize=(10, 6))

    # Graficar cada output
    for i in range(targets.shape[1]):
        plt.subplot(1, 2, i + 1)  # Crea subgráficas
        plt.scatter(targets[:, i], predicciones[:, i], alpha=0.7)
        plt.plot([-100, 100], [-100, 100], 'r--')  # Línea de
igualdad
        plt.title(f'Output {i + 1}')
        plt.xlabel('Valor Real')
        plt.ylabel('Predicción')
        plt.xlim(-100, 100)
        plt.ylim(-100, 100)
        plt.grid()

    plt.tight_layout()
    plt.show()

# Llamar a la función para graficar
graficar_predicciones()
```

```
# Mostrar el error de predicción
calcular_error()

# Crear los widgets de entrada para los datos
RentaPersonamE = widgets.FloatText(description='Renta neta/Persona
m€', value=0.0)
RentaHogarmE = widgets.FloatText(description='Renta/Hogar m€',
value=0.0)
activa = widgets.FloatText(description='% 15-64 años', value=0.0)
edupoblacion64 = widgets.FloatText(description='% población >64
años', value=0.0)
Preciomediovivienda2023mE = widgets.FloatText(description='Precio
medio vivienda 2023 m€', value=0.0)
mviviendas = widgets.FloatText(description='mViviendas', value=0.0)
Compacidad = widgets.FloatText(description='Compacidad', value=0.0)
Numviv_Numhogares = widgets.FloatText(description='Num viv/Num
hogares', value=0.0)

# Crear el botón para hacer la predicción
button = widgets.Button(description="Hacer Predicción")

# Mostrar el resultado de la predicción
output = widgets.Output()

def on_button_click(b):
    with output:
        output.clear_output()  # Limpiar el output antes de mostrar
el nuevo resultado
        result1, result2 = make_prediction(RentaPersonamE.value,
RentaHogarmE.value, activa.value, edupoblacion64.value,
Preciomediovivienda2023mE.value, mviviendas.value,
Compacidad.value, Numviv_Numhogares.value)
        print(f'Predicción (Δ Poblacional): {result1}')
        print(f'Predicción ((n0/n1)-(1/e)): {result2}')

button.on_click(on_button_click)

# Mostrar los widgets y el botón en Colab
display(RentaPersonamE, RentaHogarmE, activa, edupoblacion64,
Preciomediovivienda2023mE, mviviendas, Compacidad,
Numviv_Numhogares, button, output)
```