# Luis Maximiliano López Ramírez

# Actividad: Modulo 2 - 8. Dense + Dropout + Batch Normalization

## Problem Statement

You are a data scientist working for a school

You are asked to predict the GPA of the current students based on the following provided data:

0 StudentID int64
1 Age int64
2 Gender int64
3 Ethnicity int64
4 ParentalEducation int64
5 StudyTimeWeekly float64 6 Absences int64
7 Tutoring int64
8 ParentalSupport int64
9 Extracurricular int64
10 Sports int64
11 Music int64
12 Volunteering int64
13 GPA float64 14 GradeClass float64

The GPA is the Grade Point Average, typically ranges from 0.0 to 4.0 in most educational systems, with 4.0 representing an 'A' or excellent performance.

The minimum passing GPA can vary by institution, but it's often around 2.0. This usually corresponds to a 'C' grade, which is considered satisfactory.

You need to create a Deep Learning model capable to predict the GPA of a Student based on a set of provided features. The data provided represents 2,392 students.

In this excersice you will be requested to create a total of three models and select the most performant one.

## 1) Import Libraries

First let's import the following libraries, if there is any library that you need and is not in the list bellow feel free to include it

In [53]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

## 2) Load Data

- You will be provided with a cvs (comma separated value) file.
- You will need to add that file into a pandas dataframe, you can use the following code as reference
- The file will be available in canvas

In [54]:
```
data = pd.read_csv("Student_performance_data _.csv")
data
```

Out[54]:

| | StudentID | Age | Gender | Ethnicity | ParentalEducation | StudyTimeWeekly | Absences |
|---|---|---|---|---|---|---|---|
| **0** | 1001 | 17 | 1 | 0 | 2 | 19.833723 | 7 |
| **1** | 1002 | 18 | 0 | 0 | 1 | 15.408756 | 0 |
| **2** | 1003 | 15 | 0 | 2 | 3 | 4.210570 | 26 |
| **3** | 1004 | 17 | 1 | 0 | 3 | 10.028829 | 14 |
| **4** | 1005 | 17 | 1 | 0 | 2 | 4.672495 | 17 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **2387** | 3388 | 18 | 1 | 0 | 3 | 10.680555 | 2 |
| **2388** | 3389 | 17 | 0 | 0 | 1 | 7.583217 | 4 |
| **2389** | 3390 | 16 | 1 | 0 | 2 | 6.805500 | 20 |
| **2390** | 3391 | 16 | 1 | 1 | 0 | 12.416653 | 17 |
| **2391** | 3392 | 16 | 1 | 0 | 2 | 17.819907 | 13 |

2392 rows × 15 columns

## 3) Review you data:

Make sure you review your data. Place special attention of null or empty values.

In [55]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   StudentID          2392 non-null   int64  
 1   Age                2392 non-null   int64  
 2   Gender             2392 non-null   int64  
 3   Ethnicity          2392 non-null   int64  
 4   ParentalEducation  2392 non-null   int64  
 5   StudyTimeWeekly    2392 non-null   float64
 6   Absences           2392 non-null   int64  
 7   Tutoring           2392 non-null   int64  
 8   ParentalSupport    2392 non-null   int64  
 9   Extracurricular    2392 non-null   int64  
 10  Sports             2392 non-null   int64  
 11  Music              2392 non-null   int64  
 12  Volunteering       2392 non-null   int64  
 13  GPA                2392 non-null   float64
 14  GradeClass         2392 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB
```

## 4. Remove the columns not needed for Student performance prediction

- Choose only the columns you consider to be valuable for your model training.
- For example, StudentID might not be a good feature for your model, and thus should be removed from your main dataset, which other columns should also be removed?
- You can name that final dataset as 'dataset'

In [56]:
```
# Your code here
dataset = data.drop(['StudentID', 'Gender', 'Ethnicity'], axis=1)
```

## 5. Check if the columns has any null values:

- Here you now have your final dataset to use in your model training.
- Before moving foward review your data check for any null or empty value that might be needed to be removed

In [57]:
```
# Your code here
dataset.isnull().sum()
```

Out[57]:
```
Age                  0
ParentalEducation    0
StudyTimeWeekly      0
Absences             0
Tutoring             0
ParentalSupport      0
Extracurricular      0
Sports               0
Music                0
Volunteering         0
GPA                  0
GradeClass           0
dtype: int64
```

## 6. Prepare your data for training and for testing set:

- First create a dataset named X, with all columns but GPA. These are the features
- Next create another dataset named y, with only GPA column. This is the label
- If you go to your Imports, you will see the following import: **'from sklearn.model_selection import train_test_split'**
- Use that *train_test_split* function to create: X_train, X_test, y_train and y_test respectively. Use X and y datasets as parameters. Other parameters to use are: Test Size = 0.2, Random State = 42.
- Standarize your features (X_train and X_test) by using the StandardScaler (investigate how to use fit_transform and transform functions). This will help the training process by dealing with normilized data.

Note: Your X_train shape should be around (1913, 10). This means the dataset has 10 columns which should be the input.

In [58]:
```
# Your code here
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Crear el dataset X con todas las columnas excepto 'GPA' (las características)
X = dataset.drop(['GPA'], axis=1)

# Crear el dataset y con solo la columna 'GPA' (la etiqueta)
y = dataset['GPA']

# Dividir los datos en conjuntos de entrenamiento y prueba con train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Verificar la forma de los conjuntos para asegurarse que X_train tiene 11 columnas
print(f"X_train shape: {X_train.shape}")  # Debería ser (1913, 11)

# Estandarizar las características (X_train y X_test) usando StandardScaler
scaler = StandardScaler()
```

```
# Ajustar y transformar el conjunto de entrenamiento
X_train = scaler.fit_transform(X_train)

# Transformar el conjunto de prueba con el mismo escalador
X_test = scaler.transform(X_test)
```

```
X_train shape: (1913, 11)
```

## 7. Define your Deep Neural Network.

- This will be a Sequential Neural Network.
- With a Dense input layer with 64 units, and input dimention of 10 and Relu as the activation function.
- A Dense hidden layer with 32 units, and Relu as the activation function.
- And a Dense output layer with 1 unit, do not define an activation function so it defaults to linear, suitable for regression tasks. e.g. Dense(1)

This last part of the output layer is super important, since we want to predict the GPA, this means that we want a regression and not a classification. Linear activation function is best for regression and Sigmoid is best for Binary Classification

# Experiment 1: A single Dense Hidden Layer

In [59]:
```
# Your code here

# Definir la Red Neuronal Secuencial con una sola capa oculta
model = Sequential([
    Dense(64, input_dim=11, activation='relu'),  # Capa de entrada y oculta con 64
    Dense(1)  # Capa de salida con 1 unidad (predicción), activación lineal por def
])

# Verificar la estructura del modelo
model.summary()
```

```
c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**Model: "sequential_3"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_10 (Dense) | (None, 64) | 768 |
| dense_11 (Dense) | (None, 1) | 65 |

**Total params:** 833 (3.25 KB)

**Trainable params:** 833 (3.25 KB)

**Non-trainable params:** 0 (0.00 B)

- Capa de entrada/oculta: Se mantiene una sola capa con 64 unidades y activación relu, que actúa tanto como capa de entrada y oculta.
- Capa de salida: Tiene una unidad de salida para la predicción, con activación lineal por defecto (útil para problemas de regresión).

In [60]:
```
### 8. Compile your Neural Network

# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_

# Mostrar la arquitectura del modelo
model.summary()
```

**Model: "sequential_3"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_10 (Dense) | (None, 64) | 768 |
| dense_11 (Dense) | (None, 1) | 65 |

**Total params:** 833 (3.25 KB)

**Trainable params:** 833 (3.25 KB)

**Non-trainable params:** 0 (0.00 B)

In [61]:
```
### 9. Fit (or train) your model

# Entrenar el modelo con X_train y y_train
history = model.fit(
    X_train,              # Conjunto de entrenamiento de características
    y_train,              # Conjunto de entrenamiento de etiquetas (GPA)
    epochs=50,            # Número de iteraciones (épocas)
    batch_size=10,        # Tamaño del batch
    validation_split=0.2, # Porcentaje de datos de entrenamiento para validación
    verbose=1             # Mostrar el progreso durante el entrenamiento
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3.3751 - mean_absolute_error: 1.608
5 - val_loss: 0.2170 - val_mean_absolute_error: 0.3863
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 932us/step - loss: 0.1819 - mean_absolute_error: 0.3
426 - val_loss: 0.1179 - val_mean_absolute_error: 0.2835
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 694us/step - loss: 0.0972 - mean_absolute_error: 0.2
504 - val_loss: 0.0847 - val_mean_absolute_error: 0.2389
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 699us/step - loss: 0.0775 - mean_absolute_error: 0.2
253 - val_loss: 0.0704 - val_mean_absolute_error: 0.2178
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 0.0628 - mean_absolute_error: 0.2
026 - val_loss: 0.0621 - val_mean_absolute_error: 0.2031
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - loss: 0.0573 - mean_absolute_error: 0.1
935 - val_loss: 0.0571 - val_mean_absolute_error: 0.1948
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 0.0504 - mean_absolute_error: 0.1
843 - val_loss: 0.0515 - val_mean_absolute_error: 0.1865
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 719us/step - loss: 0.0474 - mean_absolute_error: 0.1
783 - val_loss: 0.0505 - val_mean_absolute_error: 0.1838
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 695us/step - loss: 0.0457 - mean_absolute_error: 0.1
725 - val_loss: 0.0472 - val_mean_absolute_error: 0.1786
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 0.0434 - mean_absolute_error: 0.1
682 - val_loss: 0.0467 - val_mean_absolute_error: 0.1746
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 693us/step - loss: 0.0402 - mean_absolute_error: 0.1
612 - val_loss: 0.0447 - val_mean_absolute_error: 0.1727
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 976us/step - loss: 0.0359 - mean_absolute_error: 0.1
527 - val_loss: 0.0446 - val_mean_absolute_error: 0.1711
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0348 - mean_absolute_error: 0.151
8 - val_loss: 0.0415 - val_mean_absolute_error: 0.1669
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0360 - mean_absolute_error: 0.154
0 - val_loss: 0.0425 - val_mean_absolute_error: 0.1674
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0329 - mean_absolute_error: 0.147
0 - val_loss: 0.0419 - val_mean_absolute_error: 0.1655
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0341 - mean_absolute_error: 0.148
5 - val_loss: 0.0411 - val_mean_absolute_error: 0.1637
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0331 - mean_absolute_error: 0.146
4 - val_loss: 0.0407 - val_mean_absolute_error: 0.1634
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 703us/step - loss: 0.0332 - mean_absolute_error: 0.1
468 - val_loss: 0.0442 - val_mean_absolute_error: 0.1687
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - loss: 0.0337 - mean_absolute_error: 0.1
```

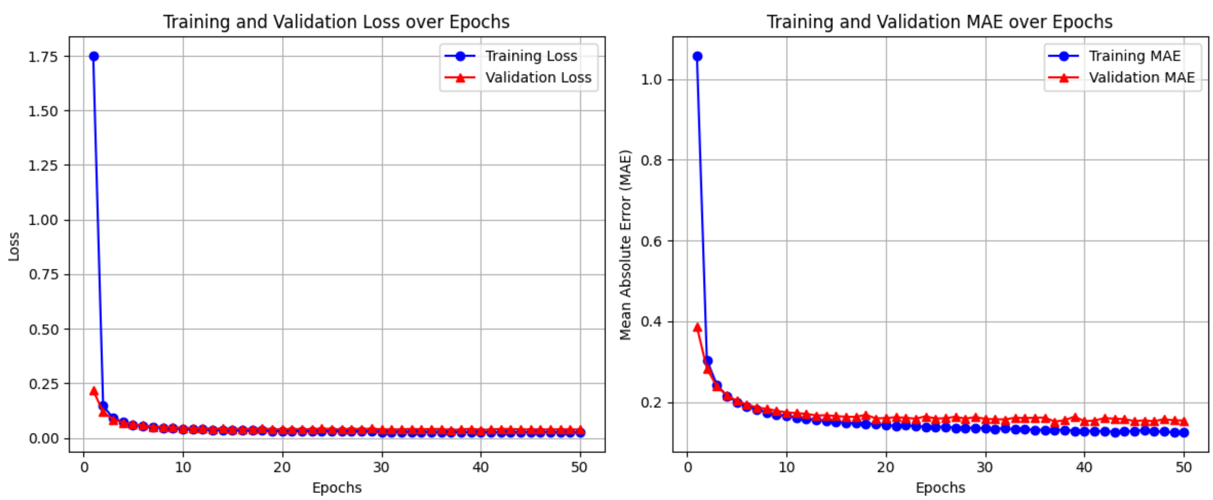
```
482 - val_loss: 0.0386 - val_mean_absolute_error: 0.1596
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 668us/step - loss: 0.0334 - mean_absolute_error: 0.1
457 - val_loss: 0.0397 - val_mean_absolute_error: 0.1605
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 0.0306 - mean_absolute_error: 0.1
414 - val_loss: 0.0403 - val_mean_absolute_error: 0.1626
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 741us/step - loss: 0.0304 - mean_absolute_error: 0.1
391 - val_loss: 0.0406 - val_mean_absolute_error: 0.1613
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 0.0316 - mean_absolute_error: 0.1
403 - val_loss: 0.0392 - val_mean_absolute_error: 0.1589
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0287 - mean_absolute_error: 0.136
2 - val_loss: 0.0432 - val_mean_absolute_error: 0.1648
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 699us/step - loss: 0.0312 - mean_absolute_error: 0.1
424 - val_loss: 0.0398 - val_mean_absolute_error: 0.1595
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - loss: 0.0289 - mean_absolute_error: 0.1
354 - val_loss: 0.0392 - val_mean_absolute_error: 0.1600
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 0.0291 - mean_absolute_error: 0.1
386 - val_loss: 0.0419 - val_mean_absolute_error: 0.1637
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step - loss: 0.0288 - mean_absolute_error: 0.1
348 - val_loss: 0.0400 - val_mean_absolute_error: 0.1590
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 743us/step - loss: 0.0276 - mean_absolute_error: 0.1
331 - val_loss: 0.0425 - val_mean_absolute_error: 0.1631
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0265 - mean_absolute_error: 0.1
313 - val_loss: 0.0393 - val_mean_absolute_error: 0.1585
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0279 - mean_absolute_error: 0.133
4 - val_loss: 0.0388 - val_mean_absolute_error: 0.1575
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 704us/step - loss: 0.0271 - mean_absolute_error: 0.1
337 - val_loss: 0.0386 - val_mean_absolute_error: 0.1571
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 707us/step - loss: 0.0256 - mean_absolute_error: 0.1
256 - val_loss: 0.0415 - val_mean_absolute_error: 0.1614
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 684us/step - loss: 0.0272 - mean_absolute_error: 0.1
297 - val_loss: 0.0393 - val_mean_absolute_error: 0.1605
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - loss: 0.0261 - mean_absolute_error: 0.1
279 - val_loss: 0.0408 - val_mean_absolute_error: 0.1614
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 0.0265 - mean_absolute_error: 0.1
308 - val_loss: 0.0408 - val_mean_absolute_error: 0.1617
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 0.0265 - mean_absolute_error: 0.1
298 - val_loss: 0.0366 - val_mean_absolute_error: 0.1518
Epoch 38/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 0.0256 - mean_absolute_error: 0.1
281 - val_loss: 0.0384 - val_mean_absolute_error: 0.1551
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 902us/step - loss: 0.0255 - mean_absolute_error: 0.1
270 - val_loss: 0.0419 - val_mean_absolute_error: 0.1641
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0294 - mean_absolute_error: 0.134
2 - val_loss: 0.0372 - val_mean_absolute_error: 0.1528
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 0.0263 - mean_absolute_error: 0.1
296 - val_loss: 0.0382 - val_mean_absolute_error: 0.1543
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0244 - mean_absolute_error: 0.123
4 - val_loss: 0.0414 - val_mean_absolute_error: 0.1604
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0258 - mean_absolute_error: 0.126
7 - val_loss: 0.0406 - val_mean_absolute_error: 0.1590
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0236 - mean_absolute_error: 0.123
3 - val_loss: 0.0393 - val_mean_absolute_error: 0.1579
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0242 - mean_absolute_error: 0.122
9 - val_loss: 0.0386 - val_mean_absolute_error: 0.1532
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0259 - mean_absolute_error: 0.127
2 - val_loss: 0.0391 - val_mean_absolute_error: 0.1546
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 891us/step - loss: 0.0231 - mean_absolute_error: 0.1
216 - val_loss: 0.0383 - val_mean_absolute_error: 0.1527
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 968us/step - loss: 0.0244 - mean_absolute_error: 0.1
244 - val_loss: 0.0398 - val_mean_absolute_error: 0.1578
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 887us/step - loss: 0.0232 - mean_absolute_error: 0.1
222 - val_loss: 0.0385 - val_mean_absolute_error: 0.1548
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 720us/step - loss: 0.0241 - mean_absolute_error: 0.1
244 - val_loss: 0.0384 - val_mean_absolute_error: 0.1545
```

In [62]:

```python
### 10. View your history variable:

# Obtener los datos del historial de entrenamiento
history_dict = history.history

# Extraer los datos para los gráficos
loss = history_dict['loss']  # Pérdida en entrenamiento
val_loss = history_dict['val_loss']  # Pérdida en validación
mae = history_dict['mean_absolute_error']  # MAE en entrenamiento
val_mae = history_dict['val_mean_absolute_error']  # MAE en validación

# Crear una lista de épocas para el eje X
epochs = range(1, len(loss) + 1)

# Gráfico 1: Pérdida de entrenamiento y validación
plt.figure(figsize=(12, 5))  # Definir el tamaño de la figura
```

```
plt.subplot(1, 2, 1)  # Crear un subplot para el primer gráfico
plt.plot(epochs, loss, 'bo-', label='Training Loss')  # 'bo-' = círculo azul para T
plt.plot(epochs, val_loss, 'r^-', label='Validation Loss')  # 'r^-' = triángulo roj
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Gráfico 2: MAE de entrenamiento y validación
plt.subplot(1, 2, 2)  # Crear un subplot para el segundo gráfico
plt.plot(epochs, mae, 'bo-', label='Training MAE')  # 'bo-' = círculo azul para Tra
plt.plot(epochs, val_mae, 'r^-', label='Validation MAE')  # 'r^-' = triángulo rojo
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Mostrar ambos gráficos
plt.tight_layout()
plt.show()
```




In [63]:
```
### 11. Evaluate your model:

# Evaluar el modelo en el conjunto de prueba
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)

# Mostrar los resultados
print(f"Test Loss: {test_loss:.4f}")
print(f"Test MAE: {test_mae:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0405 - mean_absolute_error: 0.1535
Test Loss: 0.0384
Test MAE: 0.1521
```

- Un valor bajo de test_loss indica que el modelo se ajusta bien a los datos.
- Si el MAE en el conjunto de prueba es bajo y similar al MAE de validación, el modelo generaliza bien.

- El MAE te dice cuánto se desvía en promedio el modelo del valor real. Por ejemplo, si test_mae = 0.1, significa que el modelo se desvía, en promedio, 0.1 puntos de GPA del valor real.

In [64]:
```
### 12. Use your model to make some predictions:

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Mostrar cada predicción junto con su valor real correspondiente
for i in range(10):  # Muestra las primeras 10 predicciones y valores reales
    print(f"Predicted GPA: {y_pred[i][0]:.4f} \t Actual GPA: {y_test.iloc[i]:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Predicted GPA: 1.3109    Actual GPA: 1.4277
Predicted GPA: 2.9767    Actual GPA: 3.1174
Predicted GPA: 1.9882    Actual GPA: 2.0378
Predicted GPA: 3.5744    Actual GPA: 3.5485
Predicted GPA: 0.2102    Actual GPA: 0.2490
Predicted GPA: 2.7624    Actual GPA: 2.6277
Predicted GPA: 1.6571    Actual GPA: 2.0574
Predicted GPA: 2.4684    Actual GPA: 2.2483
Predicted GPA: 2.1145    Actual GPA: 2.1947
Predicted GPA: 0.9253    Actual GPA: 0.7582
```

# Experiment 2: A set of three Dense Hidden Layers

In [65]:
```
# Definir la Red Neuronal Secuencial con tres capas densas ocultas
model = Sequential([
    Dense(128, input_dim=11, activation='relu'),  # Primera capa oculta con 128 uni
    Dense(64, activation='relu'),  # Segunda capa oculta con 64 unidades y activaci
    Dense(32, activation='relu'),  # Tercera capa oculta con 32 unidades y activaci
    Dense(1)  # Capa de salida con 1 unidad (predicción), activación lineal por def
])

# Verificar la estructura del modelo
model.summary()
```

```
c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**Model: "sequential_4"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_12 (Dense) | (None, 128) | 1,536 |
| dense_13 (Dense) | (None, 64) | 8,256 |
| dense_14 (Dense) | (None, 32) | 2,080 |
| dense_15 (Dense) | (None, 1) | 33 |

**Total params:** 11,905 (46.50 KB)

**Trainable params:** 11,905 (46.50 KB)

**Non-trainable params:** 0 (0.00 B)

- Primera capa oculta: Tiene 128 unidades y usa relu como función de activación.
- Segunda capa oculta: Tiene 64 unidades con la misma función de activación.
- Tercera capa oculta: Tiene 32 unidades y también utiliza relu.
- Capa de salida: Tiene 1 unidad para predicción, con activación lineal por defecto (adecuada para problemas de regresión).

In [66]:
```
### 8. Compile your Neural Network

# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_

# Mostrar la arquitectura del modelo
model.summary()
```

**Model: "sequential_4"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_12 (Dense) | (None, 128) | 1,536 |
| dense_13 (Dense) | (None, 64) | 8,256 |
| dense_14 (Dense) | (None, 32) | 2,080 |
| dense_15 (Dense) | (None, 1) | 33 |

**Total params:** 11,905 (46.50 KB)

**Trainable params:** 11,905 (46.50 KB)

**Non-trainable params:** 0 (0.00 B)

In [67]:
```
### 9. Fit (or train) your model

# Entrenar el modelo con X_train y y_train
history = model.fit(
    X_train,             # Conjunto de entrenamiento de características
```

```
    y_train,              # Conjunto de entrenamiento de etiquetas (GPA)
    epochs=50,            # Número de iteraciones (épocas)
    batch_size=10,        # Tamaño del batch
    validation_split=0.2, # Porcentaje de datos de entrenamiento para validación
    verbose=1             # Mostrar el progreso durante el entrenamiento
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 1.4164 - mean_absolute_error: 0.853
2 - val_loss: 0.0798 - val_mean_absolute_error: 0.2364
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0727 - mean_absolute_error: 0.218
6 - val_loss: 0.0587 - val_mean_absolute_error: 0.2004
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 890us/step - loss: 0.0538 - mean_absolute_error: 0.1
882 - val_loss: 0.0502 - val_mean_absolute_error: 0.1841
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step - loss: 0.0435 - mean_absolute_error: 0.1
657 - val_loss: 0.0491 - val_mean_absolute_error: 0.1773
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 0.0372 - mean_absolute_error: 0.1
558 - val_loss: 0.0478 - val_mean_absolute_error: 0.1721
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 0.0365 - mean_absolute_error: 0.1
530 - val_loss: 0.0462 - val_mean_absolute_error: 0.1699
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - loss: 0.0334 - mean_absolute_error: 0.1
445 - val_loss: 0.0461 - val_mean_absolute_error: 0.1699
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 0.0312 - mean_absolute_error: 0.1
394 - val_loss: 0.0450 - val_mean_absolute_error: 0.1670
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 972us/step - loss: 0.0302 - mean_absolute_error: 0.1
408 - val_loss: 0.0486 - val_mean_absolute_error: 0.1759
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0294 - mean_absolute_error: 0.134
7 - val_loss: 0.0430 - val_mean_absolute_error: 0.1645
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 0.0271 - mean_absolute_error: 0.1
307 - val_loss: 0.0427 - val_mean_absolute_error: 0.1620
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 0.0282 - mean_absolute_error: 0.1
341 - val_loss: 0.0414 - val_mean_absolute_error: 0.1599
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 0.0253 - mean_absolute_error: 0.1
268 - val_loss: 0.0458 - val_mean_absolute_error: 0.1693
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 860us/step - loss: 0.0238 - mean_absolute_error: 0.1
233 - val_loss: 0.0446 - val_mean_absolute_error: 0.1639
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0222 - mean_absolute_error: 0.119
1 - val_loss: 0.0429 - val_mean_absolute_error: 0.1642
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 949us/step - loss: 0.0225 - mean_absolute_error: 0.1
170 - val_loss: 0.0460 - val_mean_absolute_error: 0.1702
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 816us/step - loss: 0.0238 - mean_absolute_error: 0.1
221 - val_loss: 0.0471 - val_mean_absolute_error: 0.1686
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 0.0203 - mean_absolute_error: 0.1
131 - val_loss: 0.0507 - val_mean_absolute_error: 0.1781
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 0.0224 - mean_absolute_error: 0.1
```

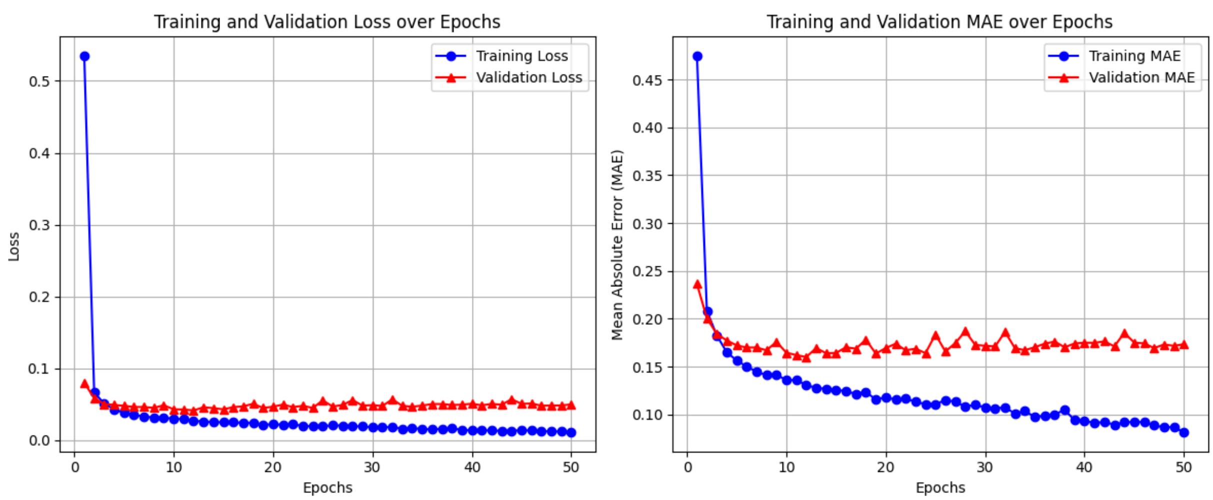
```
172 - val_loss: 0.0448 - val_mean_absolute_error: 0.1635
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 0.0197 - mean_absolute_error: 0.1
118 - val_loss: 0.0466 - val_mean_absolute_error: 0.1693
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 0.0190 - mean_absolute_error: 0.1
093 - val_loss: 0.0495 - val_mean_absolute_error: 0.1739
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 0.0217 - mean_absolute_error: 0.1
161 - val_loss: 0.0460 - val_mean_absolute_error: 0.1670
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 0.0179 - mean_absolute_error: 0.1
092 - val_loss: 0.0478 - val_mean_absolute_error: 0.1687
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - loss: 0.0175 - mean_absolute_error: 0.1
041 - val_loss: 0.0448 - val_mean_absolute_error: 0.1640
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 913us/step - loss: 0.0181 - mean_absolute_error: 0.1
056 - val_loss: 0.0554 - val_mean_absolute_error: 0.1836
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0223 - mean_absolute_error: 0.121
3 - val_loss: 0.0467 - val_mean_absolute_error: 0.1657
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 918us/step - loss: 0.0174 - mean_absolute_error: 0.1
047 - val_loss: 0.0491 - val_mean_absolute_error: 0.1745
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 0.0173 - mean_absolute_error: 0.1
031 - val_loss: 0.0558 - val_mean_absolute_error: 0.1870
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - loss: 0.0206 - mean_absolute_error: 0.1
141 - val_loss: 0.0481 - val_mean_absolute_error: 0.1726
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 0.0164 - mean_absolute_error: 0.1
026 - val_loss: 0.0488 - val_mean_absolute_error: 0.1717
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0173 - mean_absolute_error: 0.104
0 - val_loss: 0.0476 - val_mean_absolute_error: 0.1710
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 946us/step - loss: 0.0174 - mean_absolute_error: 0.1
036 - val_loss: 0.0569 - val_mean_absolute_error: 0.1862
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 0.0158 - mean_absolute_error: 0.0
996 - val_loss: 0.0478 - val_mean_absolute_error: 0.1692
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 0.0167 - mean_absolute_error: 0.1
020 - val_loss: 0.0465 - val_mean_absolute_error: 0.1671
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 0.0144 - mean_absolute_error: 0.0
943 - val_loss: 0.0481 - val_mean_absolute_error: 0.1702
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 0.0149 - mean_absolute_error: 0.0
980 - val_loss: 0.0503 - val_mean_absolute_error: 0.1739
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 0.0142 - mean_absolute_error: 0.0
942 - val_loss: 0.0502 - val_mean_absolute_error: 0.1761
Epoch 38/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 764us/step - loss: 0.0165 - mean_absolute_error: 0.1
027 - val_loss: 0.0490 - val_mean_absolute_error: 0.1702
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 805us/step - loss: 0.0129 - mean_absolute_error: 0.0
912 - val_loss: 0.0494 - val_mean_absolute_error: 0.1741
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 0.0125 - mean_absolute_error: 0.0
889 - val_loss: 0.0507 - val_mean_absolute_error: 0.1749
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 0.0121 - mean_absolute_error: 0.0
868 - val_loss: 0.0485 - val_mean_absolute_error: 0.1748
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 0.0137 - mean_absolute_error: 0.0
932 - val_loss: 0.0505 - val_mean_absolute_error: 0.1763
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 0.0122 - mean_absolute_error: 0.0
868 - val_loss: 0.0493 - val_mean_absolute_error: 0.1711
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 0.0128 - mean_absolute_error: 0.0
896 - val_loss: 0.0570 - val_mean_absolute_error: 0.1852
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 888us/step - loss: 0.0135 - mean_absolute_error: 0.0
918 - val_loss: 0.0505 - val_mean_absolute_error: 0.1751
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 0.0123 - mean_absolute_error: 0.0
878 - val_loss: 0.0511 - val_mean_absolute_error: 0.1742
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 0.0125 - mean_absolute_error: 0.0
875 - val_loss: 0.0480 - val_mean_absolute_error: 0.1693
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 0.0104 - mean_absolute_error: 0.0
797 - val_loss: 0.0482 - val_mean_absolute_error: 0.1730
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - loss: 0.0113 - mean_absolute_error: 0.0
847 - val_loss: 0.0483 - val_mean_absolute_error: 0.1716
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - loss: 0.0095 - mean_absolute_error: 0.0
776 - val_loss: 0.0498 - val_mean_absolute_error: 0.1734
```

In [68]:

```python
### 10. View your history variable:

# Obtener los datos del historial de entrenamiento
history_dict = history.history

# Extraer los datos para los gráficos
loss = history_dict['loss']  # Pérdida en entrenamiento
val_loss = history_dict['val_loss']  # Pérdida en validación
mae = history_dict['mean_absolute_error']  # MAE en entrenamiento
val_mae = history_dict['val_mean_absolute_error']  # MAE en validación

# Crear una lista de épocas para el eje X
epochs = range(1, len(loss) + 1)

# Gráfico 1: Pérdida de entrenamiento y validación
plt.figure(figsize=(12, 5))  # Definir el tamaño de la figura
```

```
plt.subplot(1, 2, 1)  # Crear un subplot para el primer gráfico
plt.plot(epochs, loss, 'bo-', label='Training Loss')  # 'bo-' = círculo azul para T
plt.plot(epochs, val_loss, 'r^-', label='Validation Loss')  # 'r^-' = triángulo roj
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Gráfico 2: MAE de entrenamiento y validación
plt.subplot(1, 2, 2)  # Crear un subplot para el segundo gráfico
plt.plot(epochs, mae, 'bo-', label='Training MAE')  # 'bo-' = círculo azul para Tra
plt.plot(epochs, val_mae, 'r^-', label='Validation MAE')  # 'r^-' = triángulo rojo
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Mostrar ambos gráficos
plt.tight_layout()
plt.show()
```




In [69]:

```
### 11. Evaluate your model:

# Evaluar el modelo en el conjunto de prueba
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)

# Mostrar los resultados
print(f"Test Loss: {test_loss:.4f}")
print(f"Test MAE: {test_mae:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0562 - mean_absolute_error: 0.1832
Test Loss: 0.0523
Test MAE: 0.1781
```

- Un valor bajo de test_loss indica que el modelo se ajusta bien a los datos.
- Si el MAE en el conjunto de prueba es bajo y similar al MAE de validación, el modelo generaliza bien.

- El MAE te dice cuánto se desvía en promedio el modelo del valor real. Por ejemplo, si test_mae = 0.1, significa que el modelo se desvía, en promedio, 0.1 puntos de GPA del valor real.

In [70]:
```
### 12. Use your model to make some predictions:

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Mostrar cada predicción junto con su valor real correspondiente
for i in range(10):  # Muestra las primeras 10 predicciones y valores reales
    print(f"Predicted GPA: {y_pred[i][0]:.4f} \t Actual GPA: {y_test.iloc[i]:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Predicted GPA: 1.5592    Actual GPA: 1.4277
Predicted GPA: 2.8664    Actual GPA: 3.1174
Predicted GPA: 2.2037    Actual GPA: 2.0378
Predicted GPA: 3.6643    Actual GPA: 3.5485
Predicted GPA: 0.6247    Actual GPA: 0.2490
Predicted GPA: 2.6479    Actual GPA: 2.6277
Predicted GPA: 1.7483    Actual GPA: 2.0574
Predicted GPA: 2.0849    Actual GPA: 2.2483
Predicted GPA: 2.2335    Actual GPA: 2.1947
Predicted GPA: 0.8945    Actual GPA: 0.7582
```

# Experiment 3: Add a dropout layer after each Dense Hidden Layer

In [71]:
```
# Your code here

# Definir la Red Neuronal Secuencial con tres capas densas ocultas y capas de Dropo
model = Sequential([
    Dense(128, input_dim=11, activation='relu'),  # Primera capa oculta
    Dropout(0.3),  # Capa de Dropout con tasa de 30%
    Dense(64, activation='relu'),  # Segunda capa oculta
    Dropout(0.3),  # Capa de Dropout con tasa de 30%
    Dense(32, activation='relu'),  # Tercera capa oculta
    Dropout(0.3),  # Capa de Dropout con tasa de 30%
    Dense(1)  # Capa de salida con 1 unidad (predicción)
])

# Verificar la estructura del modelo
model.summary()
```

```
c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape
`/`input_dim` argument to a layer. When using Sequential models, prefer using an `In
put(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**Model: "sequential_5"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_16 (Dense) | (None, 128) | 1,536 |
| dropout_3 (Dropout) | (None, 128) | 0 |
| dense_17 (Dense) | (None, 64) | 8,256 |
| dropout_4 (Dropout) | (None, 64) | 0 |
| dense_18 (Dense) | (None, 32) | 2,080 |
| dropout_5 (Dropout) | (None, 32) | 0 |
| dense_19 (Dense) | (None, 1) | 33 |

**Total params:** 11,905 (46.50 KB)

**Trainable params:** 11,905 (46.50 KB)

**Non-trainable params:** 0 (0.00 B)

- Dropout(0.3): Inserta una capa de Dropout con una tasa del 30% después de cada capa densa oculta. Esto significa que el 30% de las unidades de cada capa se desactivan aleatoriamente durante el entrenamiento para evitar el sobreajuste.
- Capas densas: Cada capa oculta tiene una función de activación relu, lo cual es común para redes profundas.
- Capa de salida: Tiene 1 unidad y la activación lineal por defecto para problemas de regresión.

In [72]:

```
### 8. Compile your Neural Network

# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_

# Mostrar la arquitectura del modelo
model.summary()
```

**Model: "sequential_5"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_16 (Dense) | (None, 128) | 1,536 |
| dropout_3 (Dropout) | (None, 128) | 0 |
| dense_17 (Dense) | (None, 64) | 8,256 |
| dropout_4 (Dropout) | (None, 64) | 0 |
| dense_18 (Dense) | (None, 32) | 2,080 |
| dropout_5 (Dropout) | (None, 32) | 0 |
| dense_19 (Dense) | (None, 1) | 33 |

**Total params:** 11,905 (46.50 KB)

**Trainable params:** 11,905 (46.50 KB)

**Non-trainable params:** 0 (0.00 B)

In [73]:
```
### 9. Fit (or train) your model

# Entrenar el modelo con X_train y y_train
history = model.fit(
    X_train,              # Conjunto de entrenamiento de características
    y_train,              # Conjunto de entrenamiento de etiquetas (GPA)
    epochs=50,            # Número de iteraciones (épocas)
    batch_size=10,        # Tamaño del batch
    validation_split=0.2, # Porcentaje de datos de entrenamiento para validación
    verbose=1             # Mostrar el progreso durante el entrenamiento
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 1.8187 - mean_absolute_error: 1.050
0 - val_loss: 0.2112 - val_mean_absolute_error: 0.3842
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.5240 - mean_absolute_error: 0.566
9 - val_loss: 0.1500 - val_mean_absolute_error: 0.3223
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3903 - mean_absolute_error: 0.489
0 - val_loss: 0.1385 - val_mean_absolute_error: 0.3133
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3458 - mean_absolute_error: 0.456
4 - val_loss: 0.0928 - val_mean_absolute_error: 0.2481
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 0.3104 - mean_absolute_error: 0.4
325 - val_loss: 0.1177 - val_mean_absolute_error: 0.2878
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2481 - mean_absolute_error: 0.389
5 - val_loss: 0.1026 - val_mean_absolute_error: 0.2628
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - loss: 0.2490 - mean_absolute_error: 0.3
864 - val_loss: 0.0676 - val_mean_absolute_error: 0.2152
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2230 - mean_absolute_error: 0.351
1 - val_loss: 0.0647 - val_mean_absolute_error: 0.2105
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2080 - mean_absolute_error: 0.356
3 - val_loss: 0.0597 - val_mean_absolute_error: 0.2006
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 0.1903 - mean_absolute_error: 0.3
371 - val_loss: 0.0762 - val_mean_absolute_error: 0.2258
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 0.1766 - mean_absolute_error: 0.3
292 - val_loss: 0.0713 - val_mean_absolute_error: 0.2190
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 0.1708 - mean_absolute_error: 0.3
128 - val_loss: 0.0764 - val_mean_absolute_error: 0.2245
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 0.1678 - mean_absolute_error: 0.3
069 - val_loss: 0.0534 - val_mean_absolute_error: 0.1869
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 0.1616 - mean_absolute_error: 0.3
088 - val_loss: 0.0539 - val_mean_absolute_error: 0.1861
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 0.1609 - mean_absolute_error: 0.3
073 - val_loss: 0.0583 - val_mean_absolute_error: 0.1951
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 0.1431 - mean_absolute_error: 0.2
933 - val_loss: 0.0928 - val_mean_absolute_error: 0.2517
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 0.1557 - mean_absolute_error: 0.3
029 - val_loss: 0.0614 - val_mean_absolute_error: 0.2005
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1362 - mean_absolute_error: 0.283
7 - val_loss: 0.0577 - val_mean_absolute_error: 0.1932
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1466 - mean_absolute_error: 0.293
```

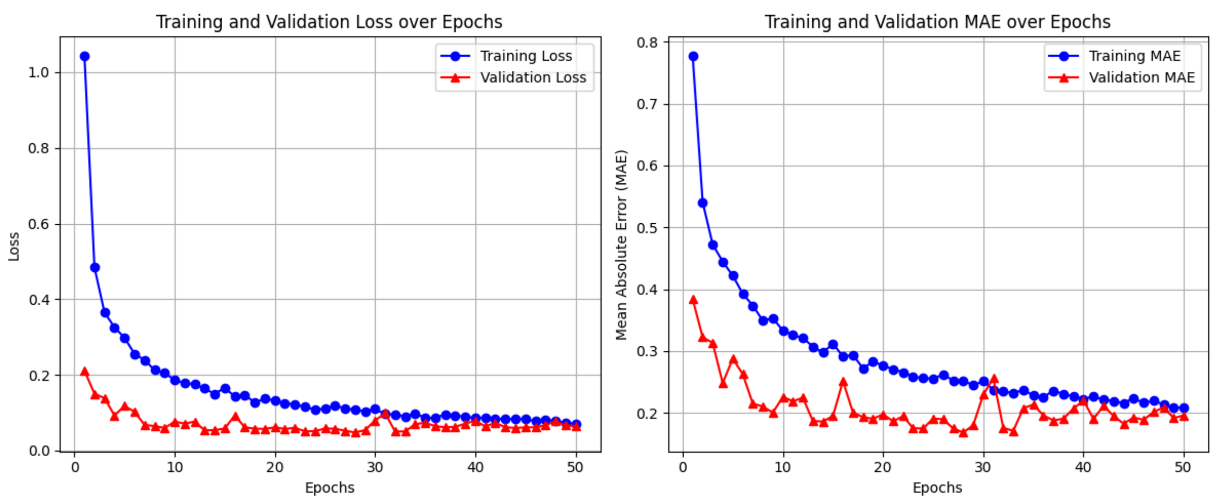
```
2 - val_loss: 0.0573 - val_mean_absolute_error: 0.1905
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 0.1469 - mean_absolute_error: 0.2
878 - val_loss: 0.0612 - val_mean_absolute_error: 0.1976
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 0.1270 - mean_absolute_error: 0.2
731 - val_loss: 0.0571 - val_mean_absolute_error: 0.1866
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 0.1181 - mean_absolute_error: 0.2
594 - val_loss: 0.0601 - val_mean_absolute_error: 0.1948
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 0.1159 - mean_absolute_error: 0.2
587 - val_loss: 0.0508 - val_mean_absolute_error: 0.1753
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 0.1037 - mean_absolute_error: 0.2
486 - val_loss: 0.0510 - val_mean_absolute_error: 0.1749
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 0.1140 - mean_absolute_error: 0.2
571 - val_loss: 0.0584 - val_mean_absolute_error: 0.1909
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 0.1327 - mean_absolute_error: 0.2
719 - val_loss: 0.0577 - val_mean_absolute_error: 0.1897
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1104 - mean_absolute_error: 0.254
2 - val_loss: 0.0524 - val_mean_absolute_error: 0.1750
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 0.1143 - mean_absolute_error: 0.2
577 - val_loss: 0.0476 - val_mean_absolute_error: 0.1684
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 0.1015 - mean_absolute_error: 0.2
467 - val_loss: 0.0540 - val_mean_absolute_error: 0.1802
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 0.1207 - mean_absolute_error: 0.2
597 - val_loss: 0.0794 - val_mean_absolute_error: 0.2295
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 0.0952 - mean_absolute_error: 0.2
340 - val_loss: 0.0987 - val_mean_absolute_error: 0.2561
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - loss: 0.0925 - mean_absolute_error: 0.2
329 - val_loss: 0.0504 - val_mean_absolute_error: 0.1757
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 0.0873 - mean_absolute_error: 0.2
267 - val_loss: 0.0503 - val_mean_absolute_error: 0.1712
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 868us/step - loss: 0.1047 - mean_absolute_error: 0.2
408 - val_loss: 0.0696 - val_mean_absolute_error: 0.2078
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0876 - mean_absolute_error: 0.228
9 - val_loss: 0.0735 - val_mean_absolute_error: 0.2134
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0862 - mean_absolute_error: 0.228
0 - val_loss: 0.0645 - val_mean_absolute_error: 0.1960
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 0.0928 - mean_absolute_error: 0.2
304 - val_loss: 0.0613 - val_mean_absolute_error: 0.1879
Epoch 38/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 0.0958 - mean_absolute_error: 0.2
331 - val_loss: 0.0626 - val_mean_absolute_error: 0.1907
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - loss: 0.0863 - mean_absolute_error: 0.2
249 - val_loss: 0.0711 - val_mean_absolute_error: 0.2071
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 0.0879 - mean_absolute_error: 0.2
237 - val_loss: 0.0787 - val_mean_absolute_error: 0.2201
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 856us/step - loss: 0.0884 - mean_absolute_error: 0.2
285 - val_loss: 0.0638 - val_mean_absolute_error: 0.1905
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 0.0870 - mean_absolute_error: 0.2
275 - val_loss: 0.0731 - val_mean_absolute_error: 0.2114
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - loss: 0.0869 - mean_absolute_error: 0.2
233 - val_loss: 0.0618 - val_mean_absolute_error: 0.1951
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 961us/step - loss: 0.0810 - mean_absolute_error: 0.2
150 - val_loss: 0.0595 - val_mean_absolute_error: 0.1826
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0816 - mean_absolute_error: 0.219
6 - val_loss: 0.0612 - val_mean_absolute_error: 0.1926
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0819 - mean_absolute_error: 0.219
2 - val_loss: 0.0614 - val_mean_absolute_error: 0.1889
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 961us/step - loss: 0.0791 - mean_absolute_error: 0.2
166 - val_loss: 0.0666 - val_mean_absolute_error: 0.2014
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 0.0759 - mean_absolute_error: 0.2
093 - val_loss: 0.0773 - val_mean_absolute_error: 0.2092
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 0.0714 - mean_absolute_error: 0.2
043 - val_loss: 0.0666 - val_mean_absolute_error: 0.1918
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 0.0752 - mean_absolute_error: 0.2
097 - val_loss: 0.0646 - val_mean_absolute_error: 0.1960
```

In [74]:

```python
### 10. View your history variable:

# Obtener los datos del historial de entrenamiento
history_dict = history.history

# Extraer los datos para los gráficos
loss = history_dict['loss']  # Pérdida en entrenamiento
val_loss = history_dict['val_loss']  # Pérdida en validación
mae = history_dict['mean_absolute_error']  # MAE en entrenamiento
val_mae = history_dict['val_mean_absolute_error']  # MAE en validación

# Crear una lista de épocas para el eje X
epochs = range(1, len(loss) + 1)

# Gráfico 1: Pérdida de entrenamiento y validación
plt.figure(figsize=(12, 5))  # Definir el tamaño de la figura
```

```
plt.subplot(1, 2, 1)  # Crear un subplot para el primer gráfico
plt.plot(epochs, loss, 'bo-', label='Training Loss')  # 'bo-' = círculo azul para T
plt.plot(epochs, val_loss, 'r^-', label='Validation Loss')  # 'r^-' = triángulo roj
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Gráfico 2: MAE de entrenamiento y validación
plt.subplot(1, 2, 2)  # Crear un subplot para el segundo gráfico
plt.plot(epochs, mae, 'bo-', label='Training MAE')  # 'bo-' = círculo azul para Tra
plt.plot(epochs, val_mae, 'r^-', label='Validation MAE')  # 'r^-' = triángulo rojo
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Mostrar ambos gráficos
plt.tight_layout()
plt.show()
```




In [75]:
```
### 11. Evaluate your model:

# Evaluar el modelo en el conjunto de prueba
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)

# Mostrar los resultados
print(f"Test Loss: {test_loss:.4f}")
print(f"Test MAE: {test_mae:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0613 - mean_absolute_error: 0.1875
Test Loss: 0.0612
Test MAE: 0.1878
```

- Un valor bajo de test_loss indica que el modelo se ajusta bien a los datos.
- Si el MAE en el conjunto de prueba es bajo y similar al MAE de validación, el modelo generaliza bien.

- El MAE te dice cuánto se desvía en promedio el modelo del valor real. Por ejemplo, si test_mae = 0.1, significa que el modelo se desvía, en promedio, 0.1 puntos de GPA del valor real.

In [76]:
```
### 12. Use your model to make some predictions:

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Mostrar cada predicción junto con su valor real correspondiente
for i in range(10):  # Muestra las primeras 10 predicciones y valores reales
    print(f"Predicted GPA: {y_pred[i][0]:.4f} \t Actual GPA: {y_test.iloc[i]:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Predicted GPA: 1.5412    Actual GPA: 1.4277
Predicted GPA: 2.9510    Actual GPA: 3.1174
Predicted GPA: 1.8327    Actual GPA: 2.0378
Predicted GPA: 3.4946    Actual GPA: 3.5485
Predicted GPA: 0.7657    Actual GPA: 0.2490
Predicted GPA: 2.6986    Actual GPA: 2.6277
Predicted GPA: 1.6716    Actual GPA: 2.0574
Predicted GPA: 2.3370    Actual GPA: 2.2483
Predicted GPA: 2.1842    Actual GPA: 2.1947
Predicted GPA: 1.1628    Actual GPA: 0.7582
```

# Experiment 4: Add a Batch Normalization Layer after each Dropout Layer.

In [77]:
```
# Your code here

# Definir la Red Neuronal Secuencial con tres capas densas ocultas y capas de Dropo
model = Sequential([
    Dense(128, input_dim=11, activation='relu'),  # Primera capa oculta
    Dropout(0.3),  # Capa de Dropout con tasa de 30%
    Dense(64, activation='relu'),  # Segunda capa oculta
    Dropout(0.3),  # Capa de Dropout con tasa de 30%
    Dense(32, activation='relu'),  # Tercera capa oculta
    Dropout(0.3),  # Capa de Dropout con tasa de 30%
    Dense(1)  # Capa de salida con 1 unidad (predicción)
])

# Verificar la estructura del modelo
model.summary()
```

```
c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**Model: "sequential_6"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_20 (Dense) | (None, 128) | 1,536 |
| dropout_6 (Dropout) | (None, 128) | 0 |
| dense_21 (Dense) | (None, 64) | 8,256 |
| dropout_7 (Dropout) | (None, 64) | 0 |
| dense_22 (Dense) | (None, 32) | 2,080 |
| dropout_8 (Dropout) | (None, 32) | 0 |
| dense_23 (Dense) | (None, 1) | 33 |

**Total params:** 11,905 (46.50 KB)

**Trainable params:** 11,905 (46.50 KB)

**Non-trainable params:** 0 (0.00 B)

In [78]:
```
### 8. Compile your Neural Network

# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_

# Mostrar la arquitectura del modelo
model.summary()
```

**Model: "sequential_6"**


| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_20 (Dense) | (None, 128) | 1,536 |
| dropout_6 (Dropout) | (None, 128) | 0 |
| dense_21 (Dense) | (None, 64) | 8,256 |
| dropout_7 (Dropout) | (None, 64) | 0 |
| dense_22 (Dense) | (None, 32) | 2,080 |
| dropout_8 (Dropout) | (None, 32) | 0 |
| dense_23 (Dense) | (None, 1) | 33 |

**Total params:** 11,905 (46.50 KB)

**Trainable params:** 11,905 (46.50 KB)

**Non-trainable params:** 0 (0.00 B)

- BatchNormalization(): Se agrega después de cada capa de Dropout para normalizar las activaciones de la capa anterior, lo cual puede ayudar a acelerar el entrenamiento y estabilizar el proceso de aprendizaje.
- Capas densas y Dropout: Las capas ocultas tienen activación relu, y las capas de Dropout ayudan a reducir el sobreajuste.

In [79]:
```
### 9. Fit (or train) your model

# Entrenar el modelo con X_train y y_train
history = model.fit(
    X_train,               # Conjunto de entrenamiento de características
    y_train,               # Conjunto de entrenamiento de etiquetas (GPA)
    epochs=50,             # Número de iteraciones (épocas)
    batch_size=10,         # Tamaño del batch
    validation_split=0.2, # Porcentaje de datos de entrenamiento para validación
    verbose=1              # Mostrar el progreso durante el entrenamiento
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 2.1000 - mean_absolute_error: 1.124
3 - val_loss: 0.1898 - val_mean_absolute_error: 0.3640
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4967 - mean_absolute_error: 0.555
9 - val_loss: 0.2861 - val_mean_absolute_error: 0.4639
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4113 - mean_absolute_error: 0.482
9 - val_loss: 0.1424 - val_mean_absolute_error: 0.3178
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 0.3484 - mean_absolute_error: 0.4
637 - val_loss: 0.1379 - val_mean_absolute_error: 0.3141
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 974us/step - loss: 0.2971 - mean_absolute_error: 0.4
282 - val_loss: 0.0853 - val_mean_absolute_error: 0.2368
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 880us/step - loss: 0.2760 - mean_absolute_error: 0.4
041 - val_loss: 0.0965 - val_mean_absolute_error: 0.2591
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2441 - mean_absolute_error: 0.383
1 - val_loss: 0.1240 - val_mean_absolute_error: 0.2944
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2463 - mean_absolute_error: 0.384
0 - val_loss: 0.1111 - val_mean_absolute_error: 0.2775
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 0.2329 - mean_absolute_error: 0.3
731 - val_loss: 0.1696 - val_mean_absolute_error: 0.3486
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 0.2157 - mean_absolute_error: 0.3
558 - val_loss: 0.0982 - val_mean_absolute_error: 0.2607
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 854us/step - loss: 0.2368 - mean_absolute_error: 0.3
753 - val_loss: 0.1062 - val_mean_absolute_error: 0.2727
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 0.1610 - mean_absolute_error: 0.3
148 - val_loss: 0.0570 - val_mean_absolute_error: 0.1939
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 0.1789 - mean_absolute_error: 0.3
256 - val_loss: 0.0475 - val_mean_absolute_error: 0.1748
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 0.1684 - mean_absolute_error: 0.3
124 - val_loss: 0.0866 - val_mean_absolute_error: 0.2427
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 0.1711 - mean_absolute_error: 0.3
170 - val_loss: 0.0570 - val_mean_absolute_error: 0.1966
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 933us/step - loss: 0.1558 - mean_absolute_error: 0.3
032 - val_loss: 0.0691 - val_mean_absolute_error: 0.2107
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 0.1646 - mean_absolute_error: 0.3
134 - val_loss: 0.0659 - val_mean_absolute_error: 0.2087
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 0.1482 - mean_absolute_error: 0.2
895 - val_loss: 0.0802 - val_mean_absolute_error: 0.2303
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 844us/step - loss: 0.1496 - mean_absolute_error: 0.2
```

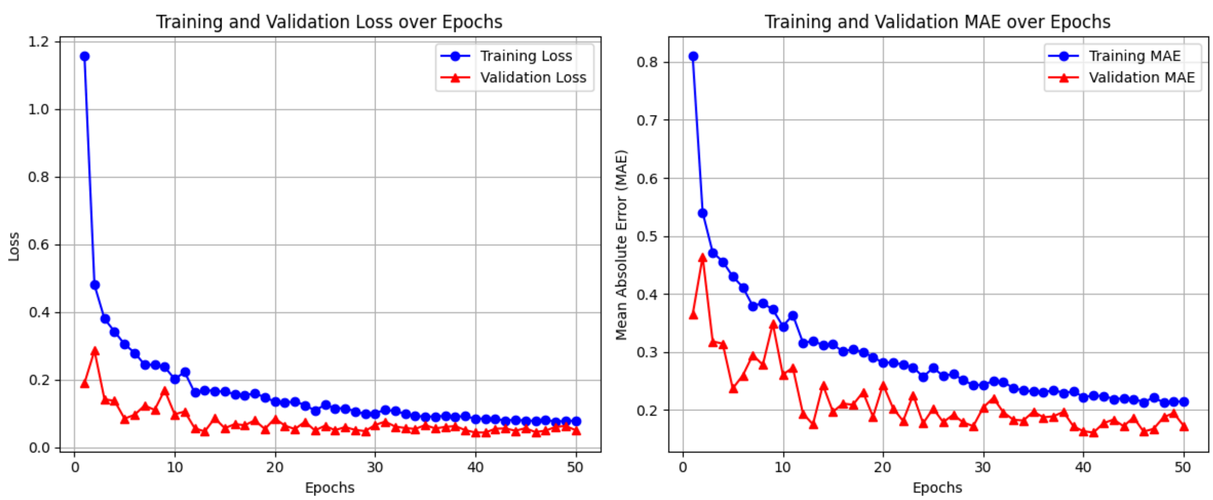
```
968 - val_loss: 0.0547 - val_mean_absolute_error: 0.1881
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 829us/step - loss: 0.1327 - mean_absolute_error: 0.2
829 - val_loss: 0.0858 - val_mean_absolute_error: 0.2435
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 0.1283 - mean_absolute_error: 0.2
787 - val_loss: 0.0647 - val_mean_absolute_error: 0.2027
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 0.1430 - mean_absolute_error: 0.2
827 - val_loss: 0.0545 - val_mean_absolute_error: 0.1806
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 0.1219 - mean_absolute_error: 0.2
677 - val_loss: 0.0761 - val_mean_absolute_error: 0.2255
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 833us/step - loss: 0.1016 - mean_absolute_error: 0.2
484 - val_loss: 0.0505 - val_mean_absolute_error: 0.1767
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 0.1279 - mean_absolute_error: 0.2
746 - val_loss: 0.0645 - val_mean_absolute_error: 0.2022
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 0.1072 - mean_absolute_error: 0.2
466 - val_loss: 0.0511 - val_mean_absolute_error: 0.1786
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 0.1138 - mean_absolute_error: 0.2
598 - val_loss: 0.0605 - val_mean_absolute_error: 0.1914
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 954us/step - loss: 0.1100 - mean_absolute_error: 0.2
536 - val_loss: 0.0522 - val_mean_absolute_error: 0.1782
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 0.1036 - mean_absolute_error: 0.2
485 - val_loss: 0.0485 - val_mean_absolute_error: 0.1718
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.1042 - mean_absolute_error: 0.2
438 - val_loss: 0.0664 - val_mean_absolute_error: 0.2041
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 0.1140 - mean_absolute_error: 0.2
524 - val_loss: 0.0766 - val_mean_absolute_error: 0.2204
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 833us/step - loss: 0.1134 - mean_absolute_error: 0.2
525 - val_loss: 0.0618 - val_mean_absolute_error: 0.1958
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 816us/step - loss: 0.0902 - mean_absolute_error: 0.2
256 - val_loss: 0.0577 - val_mean_absolute_error: 0.1828
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 870us/step - loss: 0.0949 - mean_absolute_error: 0.2
396 - val_loss: 0.0548 - val_mean_absolute_error: 0.1815
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 0.0870 - mean_absolute_error: 0.2
260 - val_loss: 0.0662 - val_mean_absolute_error: 0.1968
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 857us/step - loss: 0.0939 - mean_absolute_error: 0.2
284 - val_loss: 0.0564 - val_mean_absolute_error: 0.1870
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 0.0940 - mean_absolute_error: 0.2
361 - val_loss: 0.0608 - val_mean_absolute_error: 0.1887
Epoch 38/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 927us/step - loss: 0.0908 - mean_absolute_error: 0.2
239 - val_loss: 0.0634 - val_mean_absolute_error: 0.1964
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 0.0950 - mean_absolute_error: 0.2
339 - val_loss: 0.0513 - val_mean_absolute_error: 0.1713
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0810 - mean_absolute_error: 0.217
6 - val_loss: 0.0444 - val_mean_absolute_error: 0.1630
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0853 - mean_absolute_error: 0.224
3 - val_loss: 0.0443 - val_mean_absolute_error: 0.1608
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 882us/step - loss: 0.0854 - mean_absolute_error: 0.2
252 - val_loss: 0.0555 - val_mean_absolute_error: 0.1769
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 0.0775 - mean_absolute_error: 0.2
129 - val_loss: 0.0564 - val_mean_absolute_error: 0.1832
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 0.0888 - mean_absolute_error: 0.2
286 - val_loss: 0.0497 - val_mean_absolute_error: 0.1718
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0764 - mean_absolute_error: 0.213
3 - val_loss: 0.0571 - val_mean_absolute_error: 0.1867
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0841 - mean_absolute_error: 0.220
9 - val_loss: 0.0455 - val_mean_absolute_error: 0.1624
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 917us/step - loss: 0.0808 - mean_absolute_error: 0.2
185 - val_loss: 0.0499 - val_mean_absolute_error: 0.1672
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 0.0723 - mean_absolute_error: 0.2
079 - val_loss: 0.0607 - val_mean_absolute_error: 0.1872
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 0.0838 - mean_absolute_error: 0.2
194 - val_loss: 0.0638 - val_mean_absolute_error: 0.1944
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 0.0801 - mean_absolute_error: 0.2
139 - val_loss: 0.0514 - val_mean_absolute_error: 0.1723
```

In [80]:

```
### 10. View your history variable:

# Obtener los datos del historial de entrenamiento
history_dict = history.history

# Extraer los datos para los gráficos
loss = history_dict['loss']  # Pérdida en entrenamiento
val_loss = history_dict['val_loss']  # Pérdida en validación
mae = history_dict['mean_absolute_error']  # MAE en entrenamiento
val_mae = history_dict['val_mean_absolute_error']  # MAE en validación

# Crear una lista de épocas para el eje X
epochs = range(1, len(loss) + 1)

# Gráfico 1: Pérdida de entrenamiento y validación
plt.figure(figsize=(12, 5))  # Definir el tamaño de la figura
```

```
plt.subplot(1, 2, 1)  # Crear un subplot para el primer gráfico
plt.plot(epochs, loss, 'bo-', label='Training Loss')  # 'bo-' = círculo azul para T
plt.plot(epochs, val_loss, 'r^-', label='Validation Loss')  # 'r^-' = triángulo roj
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Gráfico 2: MAE de entrenamiento y validación
plt.subplot(1, 2, 2)  # Crear un subplot para el segundo gráfico
plt.plot(epochs, mae, 'bo-', label='Training MAE')  # 'bo-' = círculo azul para Tra
plt.plot(epochs, val_mae, 'r^-', label='Validation MAE')  # 'r^-' = triángulo rojo
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE over Epochs')
plt.legend()  # Mostrar la leyenda
plt.grid(True)  # Mostrar la cuadrícula

# Mostrar ambos gráficos
plt.tight_layout()
plt.show()
```




In [81]:
```
### 11. Evaluate your model:

# Evaluar el modelo en el conjunto de prueba
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)

# Mostrar los resultados
print(f"Test Loss: {test_loss:.4f}")
print(f"Test MAE: {test_mae:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0513 - mean_absolute_error: 0.1720
Test Loss: 0.0493
Test MAE: 0.1682
```

- Un valor bajo de test_loss indica que el modelo se ajusta bien a los datos.
- Si el MAE en el conjunto de prueba es bajo y similar al MAE de validación, el modelo generaliza bien.

- El MAE te dice cuánto se desvía en promedio el modelo del valor real. Por ejemplo, si test_mae = 0.1, significa que el modelo se desvía, en promedio, 0.1 puntos de GPA del valor real.

In [82]:
```
### 12. Use your model to make some predictions:

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Mostrar cada predicción junto con su valor real correspondiente
for i in range(10):  # Muestra las primeras 10 predicciones y valores reales
    print(f"Predicted GPA: {y_pred[i][0]:.4f} \t Actual GPA: {y_test.iloc[i]:.4f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Predicted GPA: 1.4927    Actual GPA: 1.4277
Predicted GPA: 3.0712    Actual GPA: 3.1174
Predicted GPA: 1.8588    Actual GPA: 2.0378
Predicted GPA: 3.7106    Actual GPA: 3.5485
Predicted GPA: 0.6600    Actual GPA: 0.2490
Predicted GPA: 2.7283    Actual GPA: 2.6277
Predicted GPA: 1.6210    Actual GPA: 2.0574
Predicted GPA: 2.3329    Actual GPA: 2.2483
Predicted GPA: 2.1865    Actual GPA: 2.1947
Predicted GPA: 1.1051    Actual GPA: 0.7582
```

# Tabla comparativa de los 4 experimentos

In [84]:
```
import pandas as pd

# Crear un diccionario con los resultados de cada modelo
resultados_modelos = {
    'Arquitectura': [
        'Experiment 1: A single Dense Hidden Layer',
        'Experiment 2: A set of three Dense Hidden Layers',
        'Experiment 3: Add a dropout layer after each Dense Hidden Layer',
        'Experiment 4: Add a Batch Normalization Layer after each Dropout Layer'
    ],
    'Test Loss': [
        0.0384,
        0.0523,
        0.0612,
        0.0493
    ],
    'Test MAE': [
        0.1521,
        0.1781,
        0.1878,
        0.1682
    ]
}

# Crear un DataFrame a partir del diccionario
```

```python
df_resultados = pd.DataFrame(resultados_modelos)

# Mostrar la tabla comparativa
print("Tabla Comparativa de Resultados y Arquitecturas de Modelos")
print(df_resultados)

# Visualizar el DataFrame como tabla si es necesario
df_resultados
```

```
Tabla Comparativa de Resultados y Arquitecturas de Modelos
                                     Arquitectura  Test Loss  Test MAE
0           Experiment 1: A single Dense Hidden Layer     0.0384    0.1521
1   Experiment 2: A set of three Dense Hidden Layers     0.0523    0.1781
2  Experiment 3: Add a dropout layer after each D...     0.0612    0.1878
3  Experiment 4: Add a Batch Normalization Layer ...     0.0493    0.1682
```

Out[84]:

| | Arquitectura | Test Loss | Test MAE |
|---|---|---|---|
| **0** | Experiment 1: A single Dense Hidden Layer | 0.0384 | 0.1521 |
| **1** | Experiment 2: A set of three Dense Hidden Layers | 0.0523 | 0.1781 |
| **2** | Experiment 3: Add a dropout layer after each D... | 0.0612 | 0.1878 |
| **3** | Experiment 4: Add a Batch Normalization Layer ... | 0.0493 | 0.1682 |

# Interpretación de Resultados de los Modelos

Se realizaron varios experimentos para comparar el rendimiento de distintas arquitecturas de redes neuronales en términos de **Test Loss** y **Mean Absolute Error (MAE)**. A continuación, se presenta una interpretación detallada de los resultados:

- **Modelo con una capa oculta**: Este modelo presentó el mejor rendimiento con una **Test Loss** de 0.0384 y un **MAE** de 0.1521. Esto indica que una sola capa oculta fue suficiente para capturar la complejidad de los datos sin un sobreajuste significativo.

- **Modelo con 3 capas ocultas**: La inclusión de más capas ocultas aumentó tanto la **Test Loss** (0.0523) como el **MAE** (0.1781), lo cual sugiere que el modelo pudo haber experimentado sobreajuste o que la complejidad adicional no contribuyó al aprendizaje efectivo de los datos.

- **Modelo con capas de Dropout**: Este modelo mostró un incremento en la **Test Loss** (0.0612) y el **MAE** (0.1878), lo que indica que la tasa de **Dropout** pudo haber sido demasiado alta, afectando la capacidad del modelo para aprender de manera óptima.

- **Modelo con capas de Batch Normalization**: Este modelo tuvo un rendimiento mejorado en comparación con el modelo de 3 capas ocultas, con una **Test Loss** de 0.0493 y un **MAE** de 0.1682. La normalización de lotes parece haber estabilizado el entrenamiento y reducido la pérdida, aunque no logró superar al modelo con una capa oculta.

## Conclusión

El mejor modelo fue el **modelo con una capa oculta**, ya que presentó la menor **Test Loss** y **MAE**, demostrando ser suficiente para capturar la complejidad de los datos sin necesidad de estructuras más profundas y complejas. Este resultado sugiere que, en este caso, una arquitectura más simple fue más efectiva y menos propensa al sobreajuste.