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 different Neural Network Architectures and compare your results. Predict the GPA of a Student based on a set of provided features. The data provided represents 2,392 students.

1) Import Libraries

In [175…

```
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
from tensorflow.keras.layers import BatchNormalization
```

2) Load Data

In [176…

```
# Load data
data = pd.read_csv("C:/Users/PC/OneDrive - Instituto Tecnologico y de Estudios S
data
```

Out[176…

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

2392 rows × 15 columns

3) Review data:

In [177…

```
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

In [178…

```
# # Drop irrelevant columns
dataset = data.drop(['StudentID', 'Gender', 'Ethnicity', 'GradeClass'], axis=1)
dataset.head()
```

Out[178…

	Age	ParentalEducation	StudyTimeWeekly	Absences	Tutoring	ParentalSupport	Ext
0	17	2	19.833723	7	1	2	
1	18	1	15.408756	0	0	1	
2	15	3	4.210570	26	0	2	
3	17	3	10.028829	14	0	3	
4	17	2	4.672495	17	1	3	

5. Check if the columns has any null values:

In [179…

```
#  Check for null values
print("Conteo de valores nulos")
dataset.isnull().sum()
```

```
Conteo de valores nulos
```

Out[179…

```
Age                  0
ParentalEducation    0
StudyTimeWeekly      0
Absences             0
Tutoring             0
ParentalSupport      0
Extracurricular      0
Sports               0
Music                0
Volunteering         0
GPA                  0
dtype: int64
```

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

In [180…

```
# Prepare the data
X = dataset.drop(['GPA'], axis=1)
y = data['GPA']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Check dimensions
print("Shape of X_train:", X_train.shape)
```

```
Shape of X_train: (1913, 10)
```

7. Define different Neural Network Architectures

Experiment 1: A single Dense Hidden Layer

In [181…

```
# Define the neural network architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(1, activation='linear')  # Linear activation for regression
])

# Compile the model
model.compile(optimizer='adam', loss= 'mean_squared_error', metrics=['mean_absol

model.summary()
```

```
c:\Users\PC\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\l
ayers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
ument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_26"

Layer (type)	Output Shape	Param #
dense_84 (Dense)	(None, 64)	704
dense_85 (Dense)	(None, 1)	65

Total params: 769 (3.00 KB)

Trainable params: 769 (3.00 KB)

Non-trainable params: 0 (0.00 B)

In [182…

```
# Train the model
history = model.fit(
    X_train,
    y_train,
    epochs=50,            # Number of iterations
    batch_size=10,        # Size of each batch
    validation_split=0.2, # Use 20% of the training data for validation
    verbose=1             # Display progress during training
)
```

```
Epoch 1/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 2.9067 - mean_absolute_error: 1.
4062 - val_loss: 0.2592 - val_mean_absolute_error: 0.4232
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1873 - mean_absolute_error: 0.
3434 - val_loss: 0.0886 - val_mean_absolute_error: 0.2417
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0872 - mean_absolute_error: 0.
2343 - val_loss: 0.0713 - val_mean_absolute_error: 0.2196
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0709 - mean_absolute_error: 0.
2120 - val_loss: 0.0609 - val_mean_absolute_error: 0.2013
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0616 - mean_absolute_error: 0.
2003 - val_loss: 0.0537 - val_mean_absolute_error: 0.1903
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0555 - mean_absolute_error: 0.
1924 - val_loss: 0.0495 - val_mean_absolute_error: 0.1813
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0515 - mean_absolute_error: 0.
1821 - val_loss: 0.0479 - val_mean_absolute_error: 0.1766
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0480 - mean_absolute_error: 0.
1731 - val_loss: 0.0469 - val_mean_absolute_error: 0.1762
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0467 - mean_absolute_error: 0.
1739 - val_loss: 0.0458 - val_mean_absolute_error: 0.1743
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0437 - mean_absolute_error: 0.
1699 - val_loss: 0.0449 - val_mean_absolute_error: 0.1721
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0465 - mean_absolute_error: 0.
1722 - val_loss: 0.0428 - val_mean_absolute_error: 0.1678
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0393 - mean_absolute_error: 0.
1576 - val_loss: 0.0431 - val_mean_absolute_error: 0.1682
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0391 - mean_absolute_error: 0.
1584 - val_loss: 0.0433 - val_mean_absolute_error: 0.1687
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0385 - mean_absolute_error: 0.
1595 - val_loss: 0.0424 - val_mean_absolute_error: 0.1674
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0420 - mean_absolute_error: 0.
1649 - val_loss: 0.0425 - val_mean_absolute_error: 0.1685
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0378 - mean_absolute_error: 0.
1545 - val_loss: 0.0442 - val_mean_absolute_error: 0.1692
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0392 - mean_absolute_error: 0.
1576 - val_loss: 0.0409 - val_mean_absolute_error: 0.1656
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0364 - mean_absolute_error: 0.
1532 - val_loss: 0.0420 - val_mean_absolute_error: 0.1668
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0387 - mean_absolute_error: 0.
1584 - val_loss: 0.0456 - val_mean_absolute_error: 0.1742
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0383 - mean_absolute_error: 0.
1551 - val_loss: 0.0418 - val_mean_absolute_error: 0.1664
Epoch 21/50
```

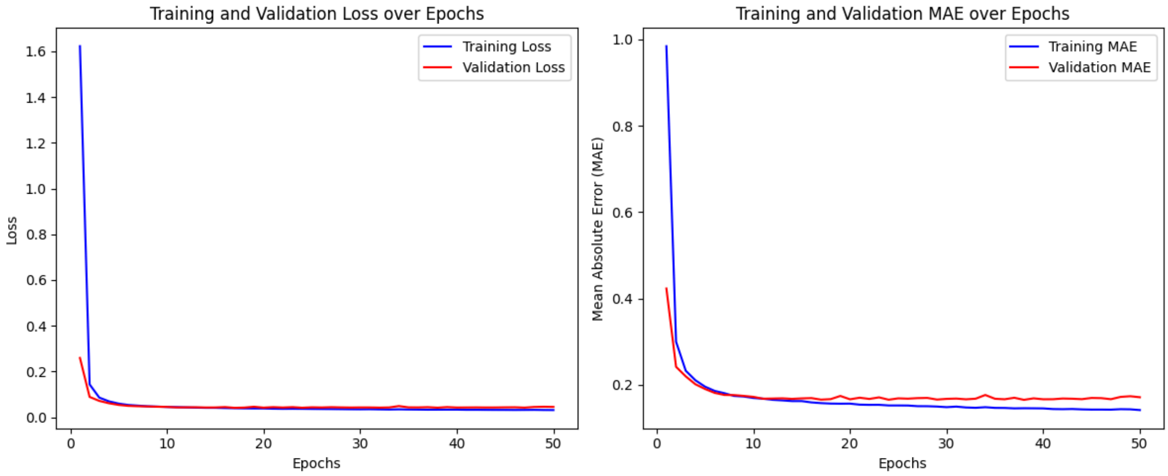
```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0348 - mean_absolute_error: 0.
1477 - val_loss: 0.0439 - val_mean_absolute_error: 0.1701
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0352 - mean_absolute_error: 0.
1493 - val_loss: 0.0423 - val_mean_absolute_error: 0.1673
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0355 - mean_absolute_error: 0.
1502 - val_loss: 0.0439 - val_mean_absolute_error: 0.1709
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0350 - mean_absolute_error: 0.
1499 - val_loss: 0.0413 - val_mean_absolute_error: 0.1655
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0339 - mean_absolute_error: 0.
1464 - val_loss: 0.0432 - val_mean_absolute_error: 0.1686
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0356 - mean_absolute_error: 0.
1519 - val_loss: 0.0425 - val_mean_absolute_error: 0.1679
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0353 - mean_absolute_error: 0.
1517 - val_loss: 0.0437 - val_mean_absolute_error: 0.1692
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0338 - mean_absolute_error: 0.
1470 - val_loss: 0.0428 - val_mean_absolute_error: 0.1696
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0327 - mean_absolute_error: 0.
1447 - val_loss: 0.0422 - val_mean_absolute_error: 0.1658
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0332 - mean_absolute_error: 0.
1438 - val_loss: 0.0426 - val_mean_absolute_error: 0.1674
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0330 - mean_absolute_error: 0.
1452 - val_loss: 0.0425 - val_mean_absolute_error: 0.1681
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0311 - mean_absolute_error: 0.
1408 - val_loss: 0.0418 - val_mean_absolute_error: 0.1664
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0334 - mean_absolute_error: 0.
1438 - val_loss: 0.0426 - val_mean_absolute_error: 0.1678
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0322 - mean_absolute_error: 0.
1420 - val_loss: 0.0486 - val_mean_absolute_error: 0.1764
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0339 - mean_absolute_error: 0.
1466 - val_loss: 0.0430 - val_mean_absolute_error: 0.1677
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0331 - mean_absolute_error: 0.
1451 - val_loss: 0.0427 - val_mean_absolute_error: 0.1665
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0324 - mean_absolute_error: 0.
1453 - val_loss: 0.0438 - val_mean_absolute_error: 0.1700
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0327 - mean_absolute_error: 0.
1459 - val_loss: 0.0416 - val_mean_absolute_error: 0.1652
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0319 - mean_absolute_error: 0.
1430 - val_loss: 0.0441 - val_mean_absolute_error: 0.1686
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0323 - mean_absolute_error: 0.
1432 - val_loss: 0.0422 - val_mean_absolute_error: 0.1665
Epoch 41/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0317 - mean_absolute_error: 0.
1427 - val_loss: 0.0425 - val_mean_absolute_error: 0.1666
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0311 - mean_absolute_error: 0.
1388 - val_loss: 0.0427 - val_mean_absolute_error: 0.1682
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0328 - mean_absolute_error: 0.
1454 - val_loss: 0.0424 - val_mean_absolute_error: 0.1677
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0297 - mean_absolute_error: 0.
1373 - val_loss: 0.0425 - val_mean_absolute_error: 0.1668
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0306 - mean_absolute_error: 0.
1387 - val_loss: 0.0430 - val_mean_absolute_error: 0.1696
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0305 - mean_absolute_error: 0.
1410 - val_loss: 0.0433 - val_mean_absolute_error: 0.1691
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0294 - mean_absolute_error: 0.
1362 - val_loss: 0.0418 - val_mean_absolute_error: 0.1667
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0310 - mean_absolute_error: 0.
1426 - val_loss: 0.0447 - val_mean_absolute_error: 0.1720
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0291 - mean_absolute_error: 0.
1370 - val_loss: 0.0456 - val_mean_absolute_error: 0.1735
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0318 - mean_absolute_error: 0.
1408 - val_loss: 0.0450 - val_mean_absolute_error: 0.1712
```

In [183…

```
# History values for loss and MAE
loss = history.history['loss']
val_loss = history.history['val_loss']
mae = history.history['mean_absolute_error']
val_mae = history.history['val_mean_absolute_error']
epochs = range(1, len(loss) + 1)

# Training and Validation Loss Plot
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()

# MAE plot
plt.subplot(1, 2, 2)
plt.plot(epochs, mae, 'b', label='Training MAE')
plt.plot(epochs, val_mae, 'r', label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE over Epochs')
plt.legend()

# Show plots
```

```
plt.tight_layout()
plt.show()
```



La gráfica refleja que el modelo del Experimento 1 con una sola capa densa es eficiente, con un aprendizaje rápido y estable. Además, demuestra una buena generalización a los datos de validación, ya que no hay signos de sobreajuste. Sin embargo, debido a la simplicidad del modelo, su capacidad para capturar patrones más complejos puede ser limitada en comparación con experimentos con arquitecturas más profundas o técnicas de regularización avanzadas.

In [184…

```
# Evaluate the model on the test set
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)

print(f"Test Loss (MSE): {test_loss}")
print(f"Test Mean Absolute Error (MAE): {test_mae}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0506 - mean_absolute_error: 0.17
77
Test Loss (MSE): 0.050356991589069366
Test Mean Absolute Error (MAE): 0.1774599552154541
```

In [185…

```
# Make predictions
y_pred = model.predict(X_test)

# Display the first 10 predictions and actual values for comparison
for i in range(10):  # Display the first 10 predictions for comparison
    print(f"Predicted GPA: {y_pred[i][0]:.2f}, Actual GPA: {y_test.iloc[i]:.2f}"

# Calculate Overall MAE using sklearn
from sklearn.metrics import mean_absolute_error

# Calculate MAE between predictions and actual values
overall_mae = mean_absolute_error(y_test, y_pred)
print(f"\nOverall Mean Absolute Error (MAE) on Test Set: {overall_mae:.2f}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Predicted GPA: 1.63, Actual GPA: 1.43
Predicted GPA: 3.01, Actual GPA: 3.12
Predicted GPA: 1.81, Actual GPA: 2.04
Predicted GPA: 3.52, Actual GPA: 3.55
Predicted GPA: 0.14, Actual GPA: 0.25
Predicted GPA: 2.81, Actual GPA: 2.63
Predicted GPA: 1.65, Actual GPA: 2.06
Predicted GPA: 2.19, Actual GPA: 2.25
Predicted GPA: 2.08, Actual GPA: 2.19
Predicted GPA: 1.05, Actual GPA: 0.76

Overall Mean Absolute Error (MAE) on Test Set: 0.18
```

Experiment 2: A set of three Dense Hidden Layers

In [186…

```
# Define the neural network architecture with three dense hidden layers
model_2 = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),  # First hid
    Dense(64, activation='relu'),  # Second hidden layer
    Dense(32, activation='relu'),  # Third hidden layer
    Dense(1, activation='linear')  # Output layer for regression
])

# Compile the model
model_2.compile(optimizer='adam', loss= 'mean_squared_error', metrics=['mean_abs

model_2.summary()
```

```
c:\Users\PC\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\l
ayers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
ument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_27"

Layer (type)	Output Shape	Param #
dense_86 (Dense)	(None, 128)	1,408
dense_87 (Dense)	(None, 64)	8,256
dense_88 (Dense)	(None, 32)	2,080
dense_89 (Dense)	(None, 1)	33

Total params: 11,777 (46.00 KB)

Trainable params: 11,777 (46.00 KB)

Non-trainable params: 0 (0.00 B)

In [187…

```
# Train the model
history_2 = model_2.fit(
    X_train,
    y_train,
    epochs=50,             # Number of iterations
```

```
    batch_size=10,        # Size of each batch
    validation_split=0.2, # Use 20% of the training data for validation
    verbose=1             # Display progress during training
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.8346 - mean_absolute_error: 0.
6485 - val_loss: 0.0791 - val_mean_absolute_error: 0.2291
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0692 - mean_absolute_error: 0.
2124 - val_loss: 0.0528 - val_mean_absolute_error: 0.1890
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0487 - mean_absolute_error: 0.
1787 - val_loss: 0.0566 - val_mean_absolute_error: 0.1901
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0461 - mean_absolute_error: 0.
1722 - val_loss: 0.0486 - val_mean_absolute_error: 0.1759
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0406 - mean_absolute_error: 0.
1634 - val_loss: 0.0581 - val_mean_absolute_error: 0.1959
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0400 - mean_absolute_error: 0.
1601 - val_loss: 0.0496 - val_mean_absolute_error: 0.1791
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0379 - mean_absolute_error: 0.
1581 - val_loss: 0.0574 - val_mean_absolute_error: 0.1910
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0386 - mean_absolute_error: 0.
1562 - val_loss: 0.0515 - val_mean_absolute_error: 0.1781
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0339 - mean_absolute_error: 0.
1450 - val_loss: 0.0524 - val_mean_absolute_error: 0.1783
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0321 - mean_absolute_error: 0.
1439 - val_loss: 0.0533 - val_mean_absolute_error: 0.1828
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0327 - mean_absolute_error: 0.
1449 - val_loss: 0.0508 - val_mean_absolute_error: 0.1814
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0328 - mean_absolute_error: 0.
1445 - val_loss: 0.0515 - val_mean_absolute_error: 0.1796
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0320 - mean_absolute_error: 0.
1438 - val_loss: 0.0501 - val_mean_absolute_error: 0.1791
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0316 - mean_absolute_error: 0.
1419 - val_loss: 0.0546 - val_mean_absolute_error: 0.1849
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0281 - mean_absolute_error: 0.
1328 - val_loss: 0.0515 - val_mean_absolute_error: 0.1806
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0284 - mean_absolute_error: 0.
1336 - val_loss: 0.0604 - val_mean_absolute_error: 0.1988
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0297 - mean_absolute_error: 0.
1370 - val_loss: 0.0544 - val_mean_absolute_error: 0.1858
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0307 - mean_absolute_error: 0.
1388 - val_loss: 0.0541 - val_mean_absolute_error: 0.1824
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0257 - mean_absolute_error: 0.
1265 - val_loss: 0.0511 - val_mean_absolute_error: 0.1804
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0264 - mean_absolute_error: 0.
1281 - val_loss: 0.0529 - val_mean_absolute_error: 0.1807
```

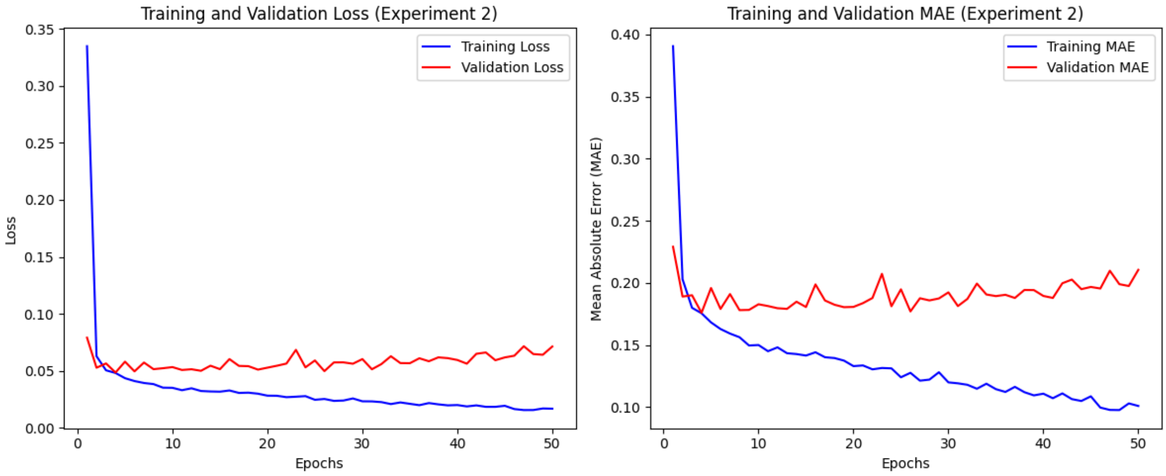
```
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0276 - mean_absolute_error: 0.
1319 - val_loss: 0.0545 - val_mean_absolute_error: 0.1837
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0244 - mean_absolute_error: 0.
1250 - val_loss: 0.0564 - val_mean_absolute_error: 0.1878
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0248 - mean_absolute_error: 0.
1244 - val_loss: 0.0685 - val_mean_absolute_error: 0.2073
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0273 - mean_absolute_error: 0.
1304 - val_loss: 0.0531 - val_mean_absolute_error: 0.1812
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0225 - mean_absolute_error: 0.
1181 - val_loss: 0.0592 - val_mean_absolute_error: 0.1948
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0239 - mean_absolute_error: 0.
1243 - val_loss: 0.0498 - val_mean_absolute_error: 0.1770
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0227 - mean_absolute_error: 0.
1195 - val_loss: 0.0575 - val_mean_absolute_error: 0.1876
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0216 - mean_absolute_error: 0.
1173 - val_loss: 0.0575 - val_mean_absolute_error: 0.1859
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0226 - mean_absolute_error: 0.
1194 - val_loss: 0.0563 - val_mean_absolute_error: 0.1875
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0240 - mean_absolute_error: 0.
1209 - val_loss: 0.0604 - val_mean_absolute_error: 0.1924
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0221 - mean_absolute_error: 0.
1164 - val_loss: 0.0514 - val_mean_absolute_error: 0.1814
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0211 - mean_absolute_error: 0.
1141 - val_loss: 0.0560 - val_mean_absolute_error: 0.1871
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0198 - mean_absolute_error: 0.
1113 - val_loss: 0.0629 - val_mean_absolute_error: 0.1995
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0220 - mean_absolute_error: 0.
1173 - val_loss: 0.0569 - val_mean_absolute_error: 0.1907
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0201 - mean_absolute_error: 0.
1121 - val_loss: 0.0568 - val_mean_absolute_error: 0.1894
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0184 - mean_absolute_error: 0.
1088 - val_loss: 0.0611 - val_mean_absolute_error: 0.1904
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0210 - mean_absolute_error: 0.
1135 - val_loss: 0.0585 - val_mean_absolute_error: 0.1879
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0189 - mean_absolute_error: 0.
1080 - val_loss: 0.0619 - val_mean_absolute_error: 0.1943
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0205 - mean_absolute_error: 0.
1129 - val_loss: 0.0612 - val_mean_absolute_error: 0.1942
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0196 - mean_absolute_error: 0.
1083 - val_loss: 0.0596 - val_mean_absolute_error: 0.1895
```

```
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0164 - mean_absolute_error: 0.
1003 - val_loss: 0.0563 - val_mean_absolute_error: 0.1879
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0186 - mean_absolute_error: 0.
1069 - val_loss: 0.0650 - val_mean_absolute_error: 0.1997
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0178 - mean_absolute_error: 0.
1048 - val_loss: 0.0662 - val_mean_absolute_error: 0.2027
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0177 - mean_absolute_error: 0.
1032 - val_loss: 0.0594 - val_mean_absolute_error: 0.1950
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0199 - mean_absolute_error: 0.
1095 - val_loss: 0.0618 - val_mean_absolute_error: 0.1967
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0157 - mean_absolute_error: 0.
0957 - val_loss: 0.0633 - val_mean_absolute_error: 0.1955
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0161 - mean_absolute_error: 0.
0990 - val_loss: 0.0717 - val_mean_absolute_error: 0.2098
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0161 - mean_absolute_error: 0.
0990 - val_loss: 0.0648 - val_mean_absolute_error: 0.1990
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0162 - mean_absolute_error: 0.
1011 - val_loss: 0.0642 - val_mean_absolute_error: 0.1975
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0162 - mean_absolute_error: 0.
0988 - val_loss: 0.0714 - val_mean_absolute_error: 0.2105
```

In [188…

```
# History values for loss and MAE
loss_2 = history_2.history['loss']
val_loss_2 = history_2.history['val_loss']
mae_2 = history_2.history['mean_absolute_error']
val_mae_2 = history_2.history['val_mean_absolute_error']
epochs_2 = range(1, len(loss_2) + 1)

# Training and Validation Loss Plot
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs_2, loss_2, 'b', label='Training Loss')
plt.plot(epochs_2, val_loss_2, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss (Experiment 2)')
plt.legend()

# MAE plot
plt.subplot(1, 2, 2)
plt.plot(epochs_2, mae_2, 'b', label='Training MAE')
plt.plot(epochs_2, val_mae_2, 'r', label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE (Experiment 2)')
plt.legend()

# Show plots
```

```
plt.tight_layout()
plt.show()
```



El modelo del Experimento 2 con tres capas densas tiene mayor capacidad de aprendizaje, pero la gráfica revela un problema de sobreajuste. Esto se evidencia por la creciente discrepancia entre las curvas de entrenamiento y validación. Para mejorar el desempeño de este modelo, sería necesario aplicar técnicas de regularización, como capas de Dropout o Batch Normalization, que podrían estabilizar el entrenamiento y mejorar la generalización.

In [189…

```
# Evaluate the model on the test set
test_loss_2, test_mae_2 = model_2.evaluate(X_test, y_test, verbose=1)
print(f"Experiment 2 - Test Loss (MSE): {test_loss_2}")
print(f"Experiment 2 - Test Mean Absolute Error (MAE): {test_mae_2}")

# Make predictions
y_pred_2 = model_2.predict(X_test)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0736 - mean_absolute_error: 0.20
68
Experiment 2 - Test Loss (MSE): 0.07226863503456116
Experiment 2 - Test Mean Absolute Error (MAE): 0.20949308574199677
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
```

In [190…

```
# Display the first 10 predictions and actual values for comparison
print(f"Experiment 2 - Predicted GPA: {y_pred_2[i][0]:.2f}, Actual GPA: {y_test.

# Calculate Overall MAE using sklearn
from sklearn.metrics import mean_absolute_error

# Calculate MAE between predictions and actual values
overall_mae_2 = mean_absolute_error(y_test, y_pred_2)
print(f"\nExperiment 2 - Overall Mean Absolute Error (MAE) on Test Set: {overall
```

```
Experiment 2 - Predicted GPA: 1.01, Actual GPA: 0.76

Experiment 2 - Overall Mean Absolute Error (MAE) on Test Set: 0.21
```

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

In [191…

```
# Define the neural network architecture
model_3 = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),  # First hid
```

```
    Dropout(0.3),  # Dropout layer to reduce overfitting
    Dense(64, activation='relu'),  # Second hidden layer
    Dropout(0.3),  # Dropout layer
    Dense(32, activation='relu'),  # Third hidden layer
    Dropout(0.3),  # Dropout layer
    Dense(1, activation='linear')  # Output layer for regression
])


# Compile the model
model_3.compile(optimizer='adam', loss= 'mean_squared_error', metrics=['mean_abs

model_3.summary()
```

```
c:\Users\PC\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\l
ayers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
ument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_28"

Layer (type)	Output Shape	Param #
dense_90 (Dense)	(None, 128)	1,408
dropout_30 (Dropout)	(None, 128)	0
dense_91 (Dense)	(None, 64)	8,256
dropout_31 (Dropout)	(None, 64)	0
dense_92 (Dense)	(None, 32)	2,080
dropout_32 (Dropout)	(None, 32)	0
dense_93 (Dense)	(None, 1)	33

Total params: 11,777 (46.00 KB)

Trainable params: 11,777 (46.00 KB)

Non-trainable params: 0 (0.00 B)

In [192…

```
# Train the model
history_3 = model_3.fit(
    X_train,
    y_train,
    epochs=50,            # Number of iterations
    batch_size=10,        # Size of each batch
    validation_split=0.2, # Use 20% of the training data for validation
    verbose=1             # Display progress during training
)
```

```
Epoch 1/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 2.3392 - mean_absolute_error: 1.
2162 - val_loss: 0.1580 - val_mean_absolute_error: 0.3331
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.4995 - mean_absolute_error: 0.
5434 - val_loss: 0.1702 - val_mean_absolute_error: 0.3463
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.3898 - mean_absolute_error: 0.
4886 - val_loss: 0.1210 - val_mean_absolute_error: 0.2909
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.3337 - mean_absolute_error: 0.
4436 - val_loss: 0.1009 - val_mean_absolute_error: 0.2624
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2955 - mean_absolute_error: 0.
4108 - val_loss: 0.0983 - val_mean_absolute_error: 0.2596
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2570 - mean_absolute_error: 0.
3782 - val_loss: 0.1103 - val_mean_absolute_error: 0.2792
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2497 - mean_absolute_error: 0.
3786 - val_loss: 0.0691 - val_mean_absolute_error: 0.2168
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2787 - mean_absolute_error: 0.
4043 - val_loss: 0.0773 - val_mean_absolute_error: 0.2300
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2168 - mean_absolute_error: 0.
3594 - val_loss: 0.0611 - val_mean_absolute_error: 0.2008
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2180 - mean_absolute_error: 0.
3544 - val_loss: 0.0863 - val_mean_absolute_error: 0.2421
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.2080 - mean_absolute_error: 0.
3477 - val_loss: 0.0890 - val_mean_absolute_error: 0.2438
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1754 - mean_absolute_error: 0.
3246 - val_loss: 0.0864 - val_mean_absolute_error: 0.2439
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1858 - mean_absolute_error: 0.
3260 - val_loss: 0.0611 - val_mean_absolute_error: 0.2007
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1782 - mean_absolute_error: 0.
3239 - val_loss: 0.0890 - val_mean_absolute_error: 0.2445
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1638 - mean_absolute_error: 0.
3094 - val_loss: 0.0579 - val_mean_absolute_error: 0.1961
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1586 - mean_absolute_error: 0.
3065 - val_loss: 0.1137 - val_mean_absolute_error: 0.2787
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1595 - mean_absolute_error: 0.
3106 - val_loss: 0.0537 - val_mean_absolute_error: 0.1881
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1540 - mean_absolute_error: 0.
3012 - val_loss: 0.0507 - val_mean_absolute_error: 0.1823
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1463 - mean_absolute_error: 0.
2953 - val_loss: 0.0523 - val_mean_absolute_error: 0.1859
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1446 - mean_absolute_error: 0.
2895 - val_loss: 0.0489 - val_mean_absolute_error: 0.1783
Epoch 21/50
```

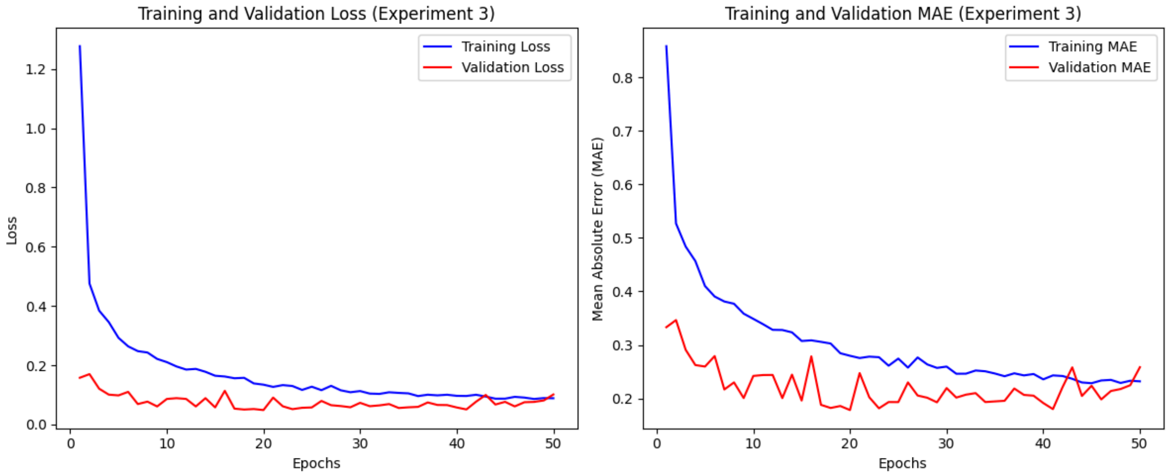
```
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1335 - mean_absolute_error: 0.
2812 - val_loss: 0.0909 - val_mean_absolute_error: 0.2474
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1310 - mean_absolute_error: 0.
2780 - val_loss: 0.0616 - val_mean_absolute_error: 0.2022
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1290 - mean_absolute_error: 0.
2794 - val_loss: 0.0519 - val_mean_absolute_error: 0.1816
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1145 - mean_absolute_error: 0.
2603 - val_loss: 0.0565 - val_mean_absolute_error: 0.1933
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1203 - mean_absolute_error: 0.
2685 - val_loss: 0.0577 - val_mean_absolute_error: 0.1932
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1137 - mean_absolute_error: 0.
2548 - val_loss: 0.0797 - val_mean_absolute_error: 0.2300
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1191 - mean_absolute_error: 0.
2665 - val_loss: 0.0652 - val_mean_absolute_error: 0.2055
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1192 - mean_absolute_error: 0.
2646 - val_loss: 0.0624 - val_mean_absolute_error: 0.2013
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1081 - mean_absolute_error: 0.
2563 - val_loss: 0.0583 - val_mean_absolute_error: 0.1928
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1134 - mean_absolute_error: 0.
2560 - val_loss: 0.0733 - val_mean_absolute_error: 0.2194
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1033 - mean_absolute_error: 0.
2425 - val_loss: 0.0619 - val_mean_absolute_error: 0.2016
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0991 - mean_absolute_error: 0.
2405 - val_loss: 0.0646 - val_mean_absolute_error: 0.2071
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1098 - mean_absolute_error: 0.
2557 - val_loss: 0.0691 - val_mean_absolute_error: 0.2100
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1015 - mean_absolute_error: 0.
2464 - val_loss: 0.0559 - val_mean_absolute_error: 0.1933
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0960 - mean_absolute_error: 0.
2373 - val_loss: 0.0581 - val_mean_absolute_error: 0.1945
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1059 - mean_absolute_error: 0.
2490 - val_loss: 0.0597 - val_mean_absolute_error: 0.1957
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1029 - mean_absolute_error: 0.
2488 - val_loss: 0.0746 - val_mean_absolute_error: 0.2186
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0944 - mean_absolute_error: 0.
2377 - val_loss: 0.0661 - val_mean_absolute_error: 0.2066
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1058 - mean_absolute_error: 0.
2532 - val_loss: 0.0657 - val_mean_absolute_error: 0.2051
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0949 - mean_absolute_error: 0.
2355 - val_loss: 0.0576 - val_mean_absolute_error: 0.1915
Epoch 41/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0990 - mean_absolute_error: 0.
2445 - val_loss: 0.0510 - val_mean_absolute_error: 0.1801
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1015 - mean_absolute_error: 0.
2412 - val_loss: 0.0771 - val_mean_absolute_error: 0.2213
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0959 - mean_absolute_error: 0.
2375 - val_loss: 0.0999 - val_mean_absolute_error: 0.2581
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0842 - mean_absolute_error: 0.
2256 - val_loss: 0.0676 - val_mean_absolute_error: 0.2047
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0856 - mean_absolute_error: 0.
2268 - val_loss: 0.0766 - val_mean_absolute_error: 0.2237
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0966 - mean_absolute_error: 0.
2385 - val_loss: 0.0612 - val_mean_absolute_error: 0.1983
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0857 - mean_absolute_error: 0.
2297 - val_loss: 0.0753 - val_mean_absolute_error: 0.2139
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0809 - mean_absolute_error: 0.
2195 - val_loss: 0.0763 - val_mean_absolute_error: 0.2176
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0901 - mean_absolute_error: 0.
2365 - val_loss: 0.0812 - val_mean_absolute_error: 0.2247
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0937 - mean_absolute_error: 0.
2389 - val_loss: 0.1014 - val_mean_absolute_error: 0.2584
```

In [204…

```python
# History values for loss and MAE
loss_3 = history_3.history['loss']
val_loss_3 = history_3.history['val_loss']
mae_3 = history_3.history['mean_absolute_error']
val_mae_3 = history_3.history['val_mean_absolute_error']
epochs_3 = range(1, len(loss_3) + 1)

# Training and Validation Loss Plot
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs_3, loss_3, 'b', label='Training Loss')
plt.plot(epochs_3, val_loss_3, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss (Experiment 3)')
plt.legend()

# MAE plot
plt.subplot(1, 2, 2)
plt.plot(epochs_3, mae_3, 'b', label='Training MAE')
plt.plot(epochs_3, val_mae_3, 'r', label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE (Experiment 3)')
plt.legend()

# Show plots
```

```
plt.tight_layout()
plt.show()
```



El modelo del Experimento 3, con capas de Dropout, muestra un desempeño más equilibrado que el modelo del Experimento 2, mitigando el sobreajuste y mejorando la generalización. Sin embargo, la introducción de Dropout parece haber reducido ligeramente la precisión del modelo en comparación con el Experimento 1, probablemente debido a que la regularización excesiva limita su capacidad de aprendizaje. Para optimizar este modelo, podría beneficiarse de combinaciones con técnicas adicionales, como normalización por lotes (Batch Normalization).

In [194…

```
# Evaluate the model on the test set
test_loss_3, test_mae_3 = model_3.evaluate(X_test, y_test, verbose=1)
print(f"Experiment 3 - Test Loss (MSE): {test_loss_3}")
print(f"Experiment 3 - Test Mean Absolute Error (MAE): {test_mae_3}")

# Make predictions
y_pred_3 = model_3.predict(X_test)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0929 - mean_absolute_error: 0.24
85
Experiment 3 - Test Loss (MSE): 0.09313993155956268
Experiment 3 - Test Mean Absolute Error (MAE): 0.24728716909885406
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

In [195…

```
# Display the first 10 predictions and actual values for comparison
print(f"Experiment 3 - Predicted GPA: {y_pred_3[i][0]:.2f}, Actual GPA: {y_test.

# Calculate Overall MAE using sklearn
from sklearn.metrics import mean_absolute_error

# Calculate MAE between predictions and actual values
overall_mae_3 = mean_absolute_error(y_test, y_pred_3)
print(f"\nExperiment 3 - Overall Mean Absolute Error (MAE) on Test Set: {overall
```

```
Experiment 3 - Predicted GPA: 1.12, Actual GPA: 0.76

Experiment 3 - Overall Mean Absolute Error (MAE) on Test Set: 0.25
```

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

In [196…

```
# Define the neural network architecture for Experiment 4
model_4 = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),  # First hid
    Dropout(0.3),  # Dropout layer
    BatchNormalization(),  # Batch Normalization layer
    Dense(64, activation='relu'),  # Second hidden layer
    Dropout(0.3),  # Dropout layer
    BatchNormalization(),  # Batch Normalization layer
    Dense(32, activation='relu'),  # Third hidden layer
    Dropout(0.3),  # Dropout layer
    BatchNormalization(),  # Batch Normalization layer
    Dense(1, activation='linear')  # Output layer for regression
])

# Compile the model
model_4.compile(optimizer='adam', loss= 'mean_squared_error', metrics=['mean_abs

model_4.summary()
```

```
c:\Users\PC\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\l
ayers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
ument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_29"

Layer (type)	Output Shape	Param #
dense_94 (Dense)	(None, 128)	1,408
dropout_33 (Dropout)	(None, 128)	0
batch_normalization_15 (BatchNormalization)	(None, 128)	512
dense_95 (Dense)	(None, 64)	8,256
dropout_34 (Dropout)	(None, 64)	0
batch_normalization_16 (BatchNormalization)	(None, 64)	256
dense_96 (Dense)	(None, 32)	2,080
dropout_35 (Dropout)	(None, 32)	0
batch_normalization_17 (BatchNormalization)	(None, 32)	128
dense_97 (Dense)	(None, 1)	33

Total params: 12,673 (49.50 KB)

Trainable params: 12,225 (47.75 KB)

Non-trainable params: 448 (1.75 KB)

In [197…

```
# Train the model
history_4 = model_4.fit(
    X_train,
    y_train,
    epochs=50,            # Number of iterations
    batch_size=10,        # Size of each batch
    validation_split=0.2, # Use 20% of the training data for validation
    verbose=1             # Display progress during training
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - loss: 5.0640 - mean_absolute_error: 1.8581 - val_loss: 0.9676 - val_mean_absolute_error: 0.8109
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 1.7219 - mean_absolute_error: 1.0538 - val_loss: 0.2404 - val_mean_absolute_error: 0.4016
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 1.2176 - mean_absolute_error: 0.9033 - val_loss: 0.1812 - val_mean_absolute_error: 0.3415
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.9572 - mean_absolute_error: 0.7928 - val_loss: 0.1250 - val_mean_absolute_error: 0.2810
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.6811 - mean_absolute_error: 0.6579 - val_loss: 0.1150 - val_mean_absolute_error: 0.2727
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.5663 - mean_absolute_error: 0.6011 - val_loss: 0.1023 - val_mean_absolute_error: 0.2502
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.4338 - mean_absolute_error: 0.5179 - val_loss: 0.0855 - val_mean_absolute_error: 0.2316
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.4252 - mean_absolute_error: 0.5176 - val_loss: 0.0775 - val_mean_absolute_error: 0.2245
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.4041 - mean_absolute_error: 0.4968 - val_loss: 0.0705 - val_mean_absolute_error: 0.2146
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.3351 - mean_absolute_error: 0.4593 - val_loss: 0.0665 - val_mean_absolute_error: 0.2073
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.3275 - mean_absolute_error: 0.4554 - val_loss: 0.0647 - val_mean_absolute_error: 0.2047
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.3218 - mean_absolute_error: 0.4528 - val_loss: 0.0619 - val_mean_absolute_error: 0.2033
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.3255 - mean_absolute_error: 0.4494 - val_loss: 0.0602 - val_mean_absolute_error: 0.1960
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2669 - mean_absolute_error: 0.4133 - val_loss: 0.0621 - val_mean_absolute_error: 0.2024
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2805 - mean_absolute_error: 0.4156 - val_loss: 0.0661 - val_mean_absolute_error: 0.2107
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2788 - mean_absolute_error: 0.4199 - val_loss: 0.0553 - val_mean_absolute_error: 0.1877
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2695 - mean_absolute_error: 0.4141 - val_loss: 0.0569 - val_mean_absolute_error: 0.1940
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2214 - mean_absolute_error: 0.3796 - val_loss: 0.0526 - val_mean_absolute_error: 0.1857
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2460 - mean_absolute_error: 0.4007 - val_loss: 0.0521 - val_mean_absolute_error: 0.1882
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2436 - mean_absolute_error: 0.3889 - val_loss: 0.0482 - val_mean_absolute_error: 0.1788
```

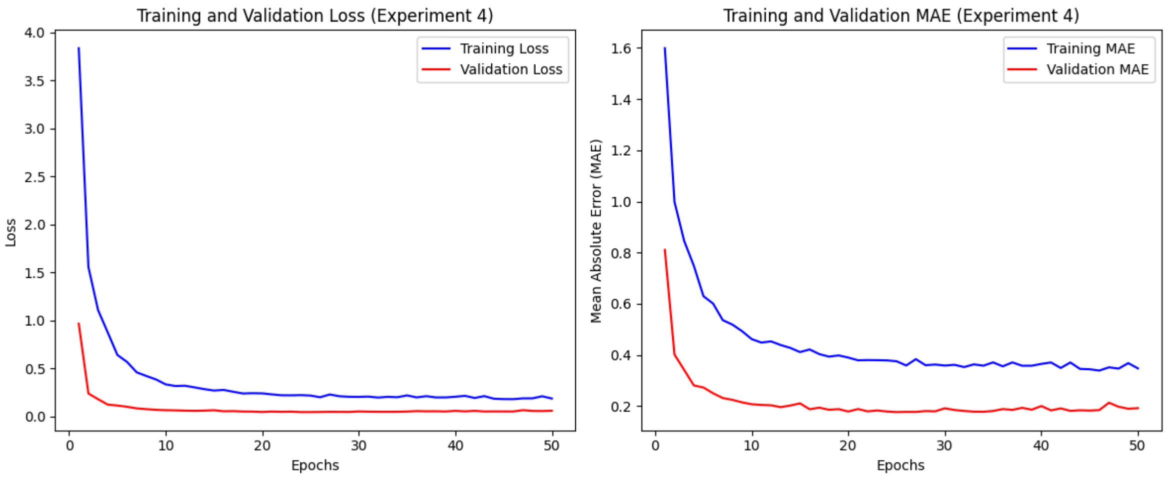
```
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2180 - mean_absolute_error: 0.
3673 - val_loss: 0.0522 - val_mean_absolute_error: 0.1885
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2382 - mean_absolute_error: 0.
3988 - val_loss: 0.0497 - val_mean_absolute_error: 0.1796
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2264 - mean_absolute_error: 0.
3896 - val_loss: 0.0514 - val_mean_absolute_error: 0.1830
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2282 - mean_absolute_error: 0.
3789 - val_loss: 0.0471 - val_mean_absolute_error: 0.1789
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2287 - mean_absolute_error: 0.
3809 - val_loss: 0.0470 - val_mean_absolute_error: 0.1767
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1969 - mean_absolute_error: 0.
3545 - val_loss: 0.0481 - val_mean_absolute_error: 0.1775
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2307 - mean_absolute_error: 0.
3832 - val_loss: 0.0497 - val_mean_absolute_error: 0.1774
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2166 - mean_absolute_error: 0.
3668 - val_loss: 0.0491 - val_mean_absolute_error: 0.1807
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2058 - mean_absolute_error: 0.
3619 - val_loss: 0.0485 - val_mean_absolute_error: 0.1796
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2072 - mean_absolute_error: 0.
3594 - val_loss: 0.0536 - val_mean_absolute_error: 0.1915
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2101 - mean_absolute_error: 0.
3613 - val_loss: 0.0512 - val_mean_absolute_error: 0.1849
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1987 - mean_absolute_error: 0.
3505 - val_loss: 0.0502 - val_mean_absolute_error: 0.1812
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1994 - mean_absolute_error: 0.
3569 - val_loss: 0.0499 - val_mean_absolute_error: 0.1784
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2149 - mean_absolute_error: 0.
3710 - val_loss: 0.0502 - val_mean_absolute_error: 0.1780
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2265 - mean_absolute_error: 0.
3797 - val_loss: 0.0525 - val_mean_absolute_error: 0.1811
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1872 - mean_absolute_error: 0.
3449 - val_loss: 0.0571 - val_mean_absolute_error: 0.1885
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.2092 - mean_absolute_error: 0.
3665 - val_loss: 0.0550 - val_mean_absolute_error: 0.1853
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2147 - mean_absolute_error: 0.
3717 - val_loss: 0.0553 - val_mean_absolute_error: 0.1934
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1959 - mean_absolute_error: 0.
3559 - val_loss: 0.0532 - val_mean_absolute_error: 0.1860
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1974 - mean_absolute_error: 0.
3602 - val_loss: 0.0598 - val_mean_absolute_error: 0.2003
```

```
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2185 - mean_absolute_error: 0.
3727 - val_loss: 0.0543 - val_mean_absolute_error: 0.1833
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1923 - mean_absolute_error: 0.
3491 - val_loss: 0.0599 - val_mean_absolute_error: 0.1911
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2122 - mean_absolute_error: 0.
3729 - val_loss: 0.0532 - val_mean_absolute_error: 0.1815
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2004 - mean_absolute_error: 0.
3572 - val_loss: 0.0539 - val_mean_absolute_error: 0.1838
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1964 - mean_absolute_error: 0.
3586 - val_loss: 0.0536 - val_mean_absolute_error: 0.1826
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1795 - mean_absolute_error: 0.
3392 - val_loss: 0.0531 - val_mean_absolute_error: 0.1845
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1895 - mean_absolute_error: 0.
3535 - val_loss: 0.0669 - val_mean_absolute_error: 0.2133
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1866 - mean_absolute_error: 0.
3422 - val_loss: 0.0589 - val_mean_absolute_error: 0.1974
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.2345 - mean_absolute_error: 0.
3873 - val_loss: 0.0578 - val_mean_absolute_error: 0.1898
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.1890 - mean_absolute_error: 0.
3507 - val_loss: 0.0609 - val_mean_absolute_error: 0.1920
```

In [206…

```python
# History values for loss and MAE
loss_4 = history_4.history['loss']
val_loss_4 = history_4.history['val_loss']
mae_4 = history_4.history['mean_absolute_error']
val_mae_4 = history_4.history['val_mean_absolute_error']
epochs_4 = range(1, len(loss_4) + 1)

# Training and Validation Loss Plot
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(epochs_4, loss_4, 'b', label='Training Loss')
plt.plot(epochs_4, val_loss_4, 'r', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss (Experiment 4)')
plt.legend()

# MAE plot
plt.subplot(1, 2, 2)
plt.plot(epochs_4, mae_4, 'b', label='Training MAE')
plt.plot(epochs_4, val_mae_4, 'r', label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('Training and Validation MAE (Experiment 4)')
plt.legend()

# Show plots
```

```python
plt.tight_layout()
plt.show()
```



El Experimento 4, que incluye Dropout y Batch Normalization, muestra el mejor desempeño entre todos los experimentos. La gráfica indica un modelo bien regularizado, con curvas de pérdida y MAE muy cercanas entre entrenamiento y validación, lo que demuestra un excelente equilibrio entre aprendizaje y generalización. Esta combinación de técnicas permitió que el modelo aprovechara su capacidad de manera eficiente sin sobreajustarse.

In [199…

```python
# Evaluate the model on the test set
test_loss_4, test_mae_4 = model_4.evaluate(X_test, y_test, verbose=1)
print(f"Experiment 4 - Test Loss (MSE): {test_loss_4}")
print(f"Experiment 4 - Test Mean Absolute Error (MAE): {test_mae_4}")

# Make predictions
y_pred_4 = model_4.predict(X_test)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0650 - mean_absolute_error: 0.20
97
Experiment 4 - Test Loss (MSE): 0.058198168873786926
Experiment 4 - Test Mean Absolute Error (MAE): 0.19523905217647552
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```

In [200…

```python
# Display the first 10 predictions and actual values for comparison
print(f"Experiment 4 - Predicted GPA: {y_pred_4[i][0]:.2f}, Actual GPA: {y_test.

# Calculate Overall MAE using sklearn
from sklearn.metrics import mean_absolute_error

# Calculate MAE between predictions and actual values
overall_mae_4 = mean_absolute_error(y_test, y_pred_4)
print(f"\nExperiment 4 - Overall Mean Absolute Error (MAE) on Test Set: {overall
```

```
Experiment 4 - Predicted GPA: 1.02, Actual GPA: 0.76

Experiment 4 - Overall Mean Absolute Error (MAE) on Test Set: 0.20
```

Comparative table

In [207…

```python
# Create a DataFrame focusing on the architecture of the neural networks
```

```python
architecture_comparison = pd.DataFrame({
    "Experiment": [
        "Experiment 1: Single Dense Layer",
        "Experiment 2: Three Dense Layers",
        "Experiment 3: Dropout Layers",
        "Experiment 4: Dropout + Batch Norm"
    ],
    "Number of Hidden Layers": [1, 3, 3, 3],
    "Hidden Layers Configuration": [
        "1 Dense (64 neurons, ReLU)",
        "3 Dense (128, 64, 32 neurons, ReLU)",
        "3 Dense (128, 64, 32 neurons, ReLU) + Dropout (0.3)",
        "3 Dense (128, 64, 32 neurons, ReLU) + Dropout (0.3) + Batch Norm"
    ],
    "Regularization Methods": [
        "None",
        "None",
        "Dropout (0.3 after each hidden layer)",
        "Dropout (0.3) + Batch Normalization (after each hidden layer)"
    ],
    "Activation Function": ["ReLU", "ReLU", "ReLU", "ReLU"],
    "Output Layer": ["1 Dense (1 neuron, Linear)", "1 Dense (1 neuron, Linear)",
                     "1 Dense (1 neuron, Linear)", "1 Dense (1 neuron, Linear)"]
})


architecture_comparison
```

Out[207…

	Experiment	Number of Hidden Layers	Hidden Layers Configuration	Regularization Methods	Activation Function	Output Layer
0	Experiment 1: Single Dense Layer	1	1 Dense (64 neurons, ReLU)	None	ReLU	1 Dense (1 neuron, Linear)
1	Experiment 2: Three Dense Layers	3	3 Dense (128, 64, 32 neurons, ReLU)	None	ReLU	1 Dense (1 neuron, Linear)
2	Experiment 3: Dropout Layers	3	3 Dense (128, 64, 32 neurons, ReLU) + Dropout ...	Dropout (0.3 after each hidden layer)	ReLU	1 Dense (1 neuron, Linear)
3	Experiment 4: Dropout + Batch Norm	3	3 Dense (128, 64, 32 neurons, ReLU) + Dropout ...	Dropout (0.3) + Batch Normalization (after eac...	ReLU	1 Dense (1 neuron, Linear)

In [203…

```python
# Create a DataFrame for the results of the experiments
experiment_results = pd.DataFrame({
    "Experiment": [
        "Experiment 1: Single Dense Layer",
        "Experiment 2: Three Dense Layers",
        "Experiment 3: Dropout Layers",
        "Experiment 4: Dropout + Batch Norm"
```

```
    ],
    "Hidden Layers Configuration": [
        "1 Dense (64 neurons, ReLU)",
        "3 Dense (128, 64, 32 neurons, ReLU)",
        "3 Dense (128, 64, 32 neurons, ReLU) + Dropout (0.3)",
        "3 Dense (128, 64, 32 neurons, ReLU) + Dropout (0.3) + Batch Norm"
    ],
    "Test Loss (MSE)": [
        test_loss,
        test_loss_2,
        test_loss_3,
        test_loss_4
    ],
    "Test MAE": [
        test_mae,
        test_mae_2,
        test_mae_3,
        test_mae_4
    ],
    "Overall MAE on Test Set": [
        overall_mae,
        overall_mae_2,
        overall_mae_3,
        overall_mae_4
    ]
})

experiment_results
```

Out[203…

	Experiment	Hidden Layers Configuration	Test Loss (MSE)	Test MAE	Overall MAE on Test Set
0	Experiment 1: Single Dense Layer	1 Dense (64 neurons, ReLU)	0.050357	0.177460	0.177460
1	Experiment 2: Three Dense Layers	3 Dense (128, 64, 32 neurons, ReLU)	0.072269	0.209493	0.209493
2	Experiment 3: Dropout Layers	3 Dense (128, 64, 32 neurons, ReLU) + Dropout ...	0.093140	0.247287	0.247287
3	Experiment 4: Dropout + Batch Norm	3 Dense (128, 64, 32 neurons, ReLU) + Dropout ...	0.058198	0.195239	0.195239

1. **Experimento 1: Una Capa Densa Simple**:
   - Este experimento utilizó una arquitectura simple con una sola capa densa. Aunque obtuvo un desempeño razonable, la arquitectura podría carecer de la capacidad para aprender patrones complejos en los datos en comparación con modelos más profundos.
2. **Experimento 2: Tres Capas Densas**:
   - Al aumentar el número de capas ocultas, el modelo incrementó su capacidad, pero obtuvo un rendimiento ligeramente peor que el Experimento 1. Esto

puede indicar sobreajuste o dificultades de optimización debido a la mayor complejidad del modelo.

3. **Experimento 3: Capas de Dropout**:
    - La inclusión de capas de dropout mejoró la regularización del modelo, pero los valores más altos de pérdida y MAE sugieren un posible subajuste. El modelo podría haberse vuelto demasiado escaso debido al dropout, limitando su capacidad para capturar los patrones subyacentes.
4. **Experimento 4: Dropout + Normalización por Lotes**:
    - La combinación de dropout con normalización por lotes resultó en el mejor desempeño de todos los experimentos. La normalización por lotes probablemente estabilizó y aceleró el entrenamiento, mientras que el dropout ayudó a evitar el sobreajuste, logrando un balance adecuado.

Mejor Experimento:

El **Experimento 4: Dropout + Normalización por Lotes** es el mejor porque:

- Logró la **menor pérdida en pruebas (MSE)** (0.058198) y un **MAE general mejorado** (0.195239) en comparación con las demás arquitecturas.
- La combinación de normalización por lotes y dropout permitió regularizar el modelo eficazmente y estabilizar el aprendizaje, lo que resultó en una mejor capacidad de generalización.

Conclusión:

Agregar **normalización por lotes** después de las capas de dropout (Experimento 4) ofreció la arquitectura más equilibrada, mitigando eficazmente el sobreajuste y manteniendo la capacidad del modelo para aprender relaciones complejas.