Problem Statement

Continuing with the same scenario, now that you have been able to successfuly predict each student GPA, now you will classify each Student based on they probability to have a successful GPA score.

The different classes are:

- Low : Students where final GPA is predicted to be between: 0 and 2
- Medium : Students where final GPA is predicted to be between: 2 and 3.5
- High : Students where final GPA is predicted to be between: 3.5 and 5

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 [2]:
```
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, LabelEncoder
import seaborn as sns
```

2) Load Data

- You will use the same file from the previous activity (Student Performance Data)

In [4]:
```
data = pd.read_csv("Student_performance_data _.csv")
data
```

Out[4]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Abser
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

In [ ]:

```
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
```

3) Add a new column called 'Profile' this column will have the following information

Based on the value of GPA for each student:

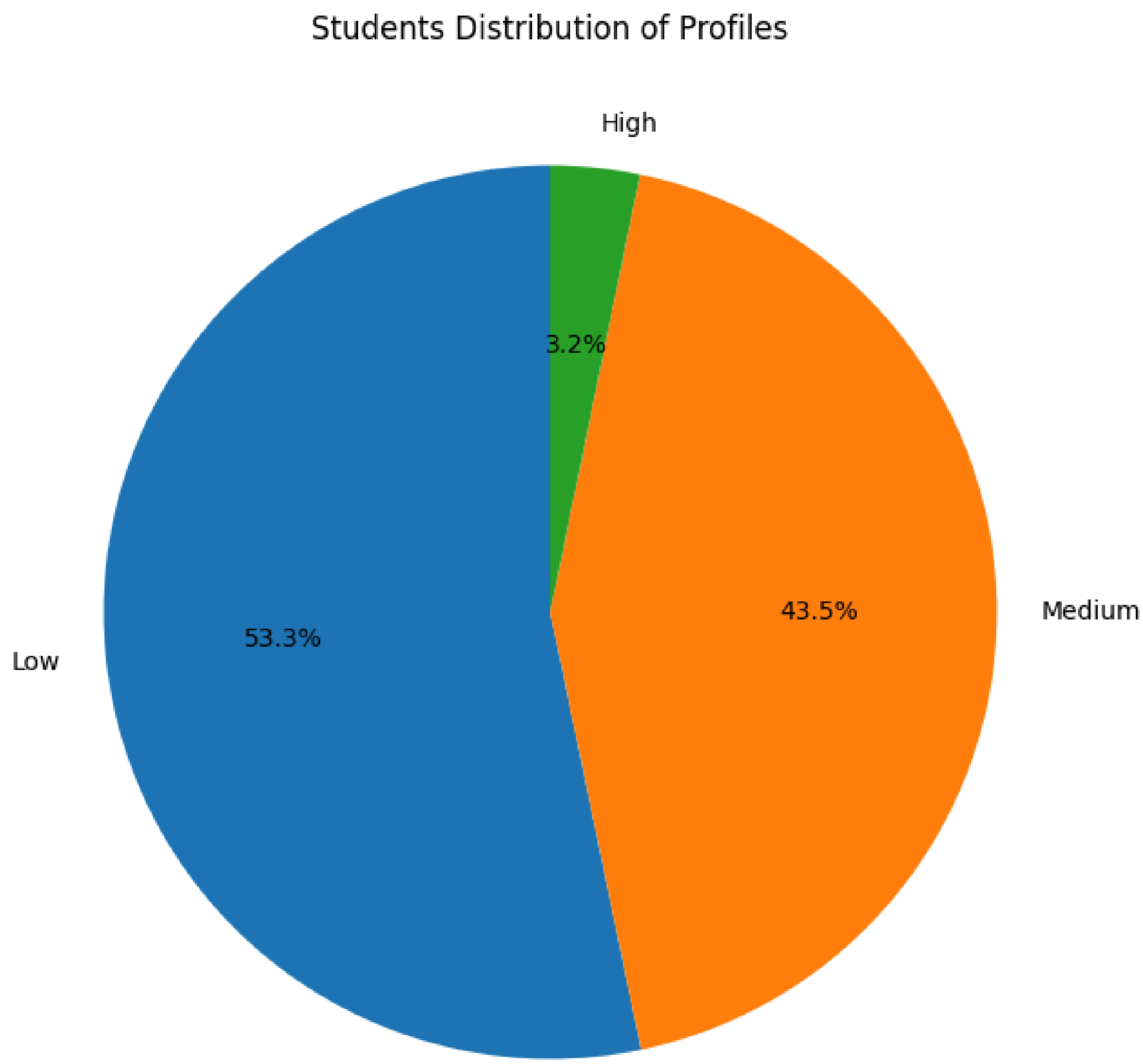
- If GPA values between 0 and 2 will be labeled 'Low',
- Values between 2 and 3.5 will be 'Medium',
- And values between 3.5 and 5 will be 'High'.

In [ ]:
```
def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)
```

4) Use Matplotlib to show a Pie chart to show the percentage of students in each profile.

- Title: Students distribution of Profiles
- Graph Type: pie

In [ ]:
```
import matplotlib.pyplot as plt
profile_counts = data['Profile'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(profile_counts, labels=profile_counts.index, autopct='%1.1f%%', star
plt.title('Students Distribution of Profiles')
plt.show()
```



5) Convert the Profile column into a Categorical Int

You have already created a column with three different values: 'Low', 'Medium', 'High'. These are Categorical values. But, it is important to notice that Neural Networks works better with numbers, since we apply mathematical operations to them.

Next you need to convert Profile values from Low, Medium and High, to 0, 1 and 2. IMPORTANT, the order does not matter, but make sure you always assign the same number to Low, same number to Medium and same number to High.

Make sure to use the fit_transform method from LabelEncoder.

In [ ]:
```
label_encoder = LabelEncoder()

data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])
data
```

Out[ ]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Abse
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 × 17 columns

6) Select the columns for your model.

Same as the last excersice we need a dataset for features and a dataset for label.

- Create the following dataset:
  - A dataset with the columns for the model.
  - From that data set generate the 'X' dataset. This dataset will have all the features (make sure Profile is NOT in this dataset)
  - Generate a second 'y' dataset, This dataset will only have our label column, which is 'Profile'.
  - Generate the Train and Test datasets for each X and y:
    - X_train with 80% of the data
    - X_test with 20% of the data
    - y_train with 80% of the data
    - y_test with 20% of the data

In [ ]:

```
data = data.drop(columns=['GradeClass', 'StudentID', 'ParentalSupport', 'Eth

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

print(f"\nTamaño de X_train: {X_train.shape}")
print(f"Tamaño de X_test: {X_test.shape}")
```

```
print(f"Tamaño de y_train: {y_train.shape}")
print(f"Tamaño de y_test: {y_test.shape}")
```

```
Tamaño de X_train: (1913, 11)
Tamaño de X_test: (479, 11)
Tamaño de y_train: (1913,)
Tamaño de y_test: (479,)
```

7) All Feature datasets in the same scale.

Use StandardScaler to make sure all features in the X_train and X_test datasets are on the same scale.

Standardization transforms your data so that it has a mean of 0 and a standard deviation of 1. This is important because many machine learning algorithms perform better when the input features are on a similar scale.

Reason for Using StandardScaler:

- Consistent Scale: Features with different scales (e.g., age in years, income in dollars) can bias the model. StandardScaler ensures all features contribute equally.
- Improved Convergence: Algorithms like gradient descent converge faster with standardized data.
- Regularization: Helps in achieving better performance in regularization methods like Ridge and Lasso regression.

In [ ]:
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train.shape
```

Out[ ]:
```
(1913, 11)
```

8. Define your Deep Neural Network.

- This will be a Sequential Neural Network.
- With a Dense input layer with 64 units, and input dimention based on the X_train size 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 the number of different values in the y dataset, activation function = to sofmax

This last part of the output layer is super important, since we want to do a classification and not a regression, we will use activation functions that fits better a classification scenario.

In [ ]:
```
model = Sequential()
model.add(Dense(64, input_dim=11, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

9. Compile your Neural Network

- Choose Adam as the optimizer
- And sparse_categorical_crossentropy as the Loss function
- Also add the following metrics: accuracy

In [ ]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metr
```

10. Fit (or train) your model

- Use the X_train and y_train datasets for the training
- Do 50 data iterations
- Choose the batch size = 10
- Also select a validation_split of 0.2
- Save the result of the fit function in a variable called 'history'

In [ ]:
```
history = model.fit(X_train, y_train, epochs=50, batch_size=10, validation_s
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7404 - loss: 0.7501 -
val_accuracy: 0.9347 - val_loss: 0.2830
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9216 - loss: 0.2707 -
val_accuracy: 0.9426 - val_loss: 0.1961
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9389 - loss: 0.1887 -
val_accuracy: 0.9478 - val_loss: 0.1502
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9529 - loss: 0.1322 -
val_accuracy: 0.9556 - val_loss: 0.1230
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9531 - loss: 0.1191 -
val_accuracy: 0.9608 - val_loss: 0.1114
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9710 - loss: 0.0865 -
val_accuracy: 0.9634 - val_loss: 0.0935
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9618 - loss: 0.0954 -
val_accuracy: 0.9713 - val_loss: 0.0805
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9807 - loss: 0.0603 -
val_accuracy: 0.9739 - val_loss: 0.0734
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9813 - loss: 0.0566 -
val_accuracy: 0.9791 - val_loss: 0.0643
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9878 - loss: 0.0466 -
val_accuracy: 0.9791 - val_loss: 0.0582
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9834 - loss: 0.0452 -
val_accuracy: 0.9791 - val_loss: 0.0600
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9958 - loss: 0.0376 -
val_accuracy: 0.9843 - val_loss: 0.0522
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9941 - loss: 0.0343 -
val_accuracy: 0.9739 - val_loss: 0.0566
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9957 - loss: 0.0280 -
val_accuracy: 0.9791 - val_loss: 0.0473
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9936 - loss: 0.0289 -
val_accuracy: 0.9817 - val_loss: 0.0469
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9963 - loss: 0.0219 -
val_accuracy: 0.9791 - val_loss: 0.0519
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9853 - loss: 0.0307 -
val_accuracy: 0.9843 - val_loss: 0.0430
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9953 - loss: 0.0179 -
val_accuracy: 0.9843 - val_loss: 0.0365
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9988 - loss: 0.0146 -
```

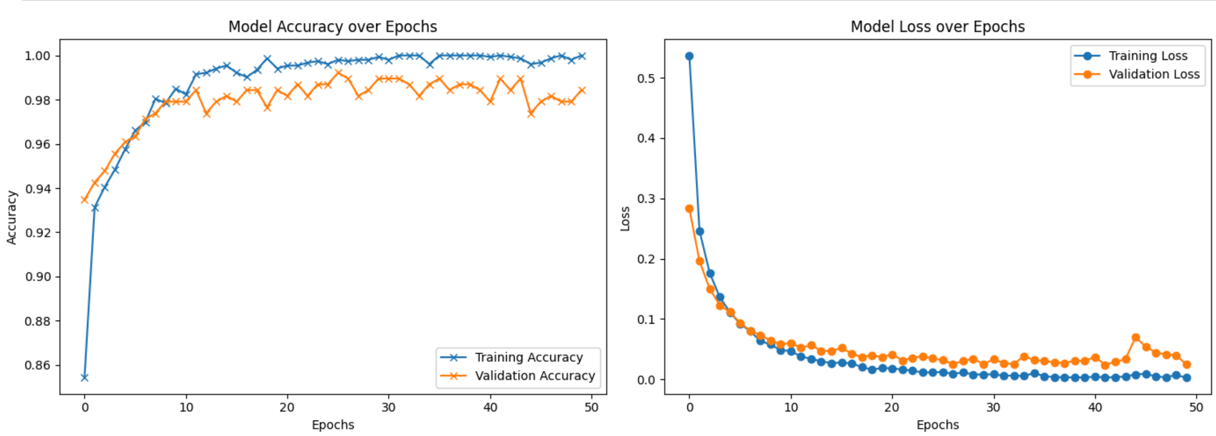
```
val_accuracy: 0.9765 - val_loss: 0.0399
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9973 - loss: 0.0163 -
val_accuracy: 0.9843 - val_loss: 0.0366
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9946 - loss: 0.0217 -
val_accuracy: 0.9817 - val_loss: 0.0412
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9949 - loss: 0.0164 -
val_accuracy: 0.9869 - val_loss: 0.0314
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9987 - loss: 0.0116 -
val_accuracy: 0.9817 - val_loss: 0.0355
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9989 - loss: 0.0107 -
val_accuracy: 0.9869 - val_loss: 0.0380
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9972 - loss: 0.0113 -
val_accuracy: 0.9869 - val_loss: 0.0344
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9975 - loss: 0.0116 -
val_accuracy: 0.9922 - val_loss: 0.0316
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9972 - loss: 0.0089 -
val_accuracy: 0.9896 - val_loss: 0.0252
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9961 - loss: 0.0136 -
val_accuracy: 0.9817 - val_loss: 0.0303
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9973 - loss: 0.0077 -
val_accuracy: 0.9843 - val_loss: 0.0343
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9991 - loss: 0.0081 -
val_accuracy: 0.9896 - val_loss: 0.0247
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9993 - loss: 0.0074 -
val_accuracy: 0.9896 - val_loss: 0.0334
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0046 -
val_accuracy: 0.9896 - val_loss: 0.0266
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0066 -
val_accuracy: 0.9869 - val_loss: 0.0255
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0055 -
val_accuracy: 0.9817 - val_loss: 0.0385
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9942 - loss: 0.0141 -
val_accuracy: 0.9869 - val_loss: 0.0316
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0042 -
val_accuracy: 0.9896 - val_loss: 0.0306
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0036 -
val_accuracy: 0.9843 - val_loss: 0.0279
Epoch 38/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0037 -
val_accuracy: 0.9869 - val_loss: 0.0270
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0033 -
val_accuracy: 0.9869 - val_loss: 0.0307
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0028 -
val_accuracy: 0.9843 - val_loss: 0.0304
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9994 - loss: 0.0044 -
val_accuracy: 0.9791 - val_loss: 0.0370
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0038 -
val_accuracy: 0.9896 - val_loss: 0.0241
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9995 - loss: 0.0024 -
val_accuracy: 0.9843 - val_loss: 0.0289
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9997 - loss: 0.0042 -
val_accuracy: 0.9896 - val_loss: 0.0333
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9982 - loss: 0.0050 -
val_accuracy: 0.9739 - val_loss: 0.0695
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9939 - loss: 0.0127 -
val_accuracy: 0.9791 - val_loss: 0.0544
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9986 - loss: 0.0047 -
val_accuracy: 0.9817 - val_loss: 0.0440
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0036 -
val_accuracy: 0.9791 - val_loss: 0.0413
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9993 - loss: 0.0038 -
val_accuracy: 0.9791 - val_loss: 0.0399
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0026 -
val_accuracy: 0.9843 - val_loss: 0.0250
```

11. View your history variable:

- Use Matplotlib.pyplot to show graphs of your model traning history
- In one graph:
  - Plot the Training Accuracy and the Validation Accuracy
  - X Label = Epochs
  - Y Label = Accuracy
  - Title = Model Accuracy over Epochs
- In a second graph:
  - Plot the Training Loss and the Validation Loss
  - X Label = Epochs
  - Y Label = Loss
  - Title = Model Loss over Epochs

In [ ]:
```
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

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



12. Evaluate your model:

- See the result of your loss function.
- What can you deduct from there?

In [ ]:
```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Loss: {loss}")
print(f"Accuracy: {accuracy}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9695 - loss: 0.0961
Loss: 0.137844979763031
Accuracy: 0.9665970802307129
```

13. Use your model to make some predictions:

- Make predictions of your X_test dataset
- Print the each of the predictions and the actual value (which is in y_test)
- Replace the 'Low', 'Medium' and 'High' to your actual and predicted values.
- How good was your model?

In [ ]:
```
from sklearn.metrics import classification_report
y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)

profile_mapping = {0: 'Low', 1: 'Medium', 2: 'High'}
y_pred_profiles = [profile_mapping[label] for label in y_pred_labels]
y_test_profiles = [profile_mapping[label] for label in y_test]

print(classification_report(y_test, y_pred_labels))
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
              precision    recall  f1-score   support

           0       0.92      0.69      0.79        16
           1       0.98      0.98      0.98       249
           2       0.96      0.97      0.96       214

    accuracy                           0.97       479
   macro avg       0.95      0.88      0.91       479
weighted avg       0.97      0.97      0.97       479
```

14. Compete against this model:

- Create two more different models to compete with this model
- Here are a few ideas of things you can change:
  - During Dataset data engineering:
    - You can remove features that you think do not help in the training and prediction
    - Feature Scaling: Ensure all features are on a similar scale (as you already did with StandardScaler)
  - During Model Definition:

- You can change the Model Architecture (change the type or number of layers or the number of units)
- You can add dropout layers to prevent overfitting
  - During Model Compile:
    - You can try other optimizer when compiling your model, here some optimizer samples: Adam, RMSprop, or Adagrad.
    - Try another Loss Function
  - During Model Training:
    - Encrease the number of Epochs
    - Adjust the size of your batch
- Explain in a Markdown cell which changes are you implementing
- Show the comparison of your model versus the original model

Model 2:

- Changes:
  - Dataset Data Engineering
  - Model Definition
  - Model Compile
  - Model Training

In [ ]:
```
data = pd.read_csv("Student_performance_data _.csv")
def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)

label_encoder = LabelEncoder()
data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])

data = data.drop(columns=['StudentID', 'GradeClass', 'Ethnicity', 'Gender'])

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, rar



scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
model2 = Sequential()
model2.add(Dense(64, input_dim=11, activation='relu'))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(3, activation='softmax'))
model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

history = model2.fit(X_train, y_train, epochs=50, batch_size=10, validation_
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model 2 Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model 2 Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

loss2, accuracy2 = model2.evaluate(X_test, y_test)
print(f"Loss: {loss2}")
print(f"Accuracy: {accuracy2}")
```

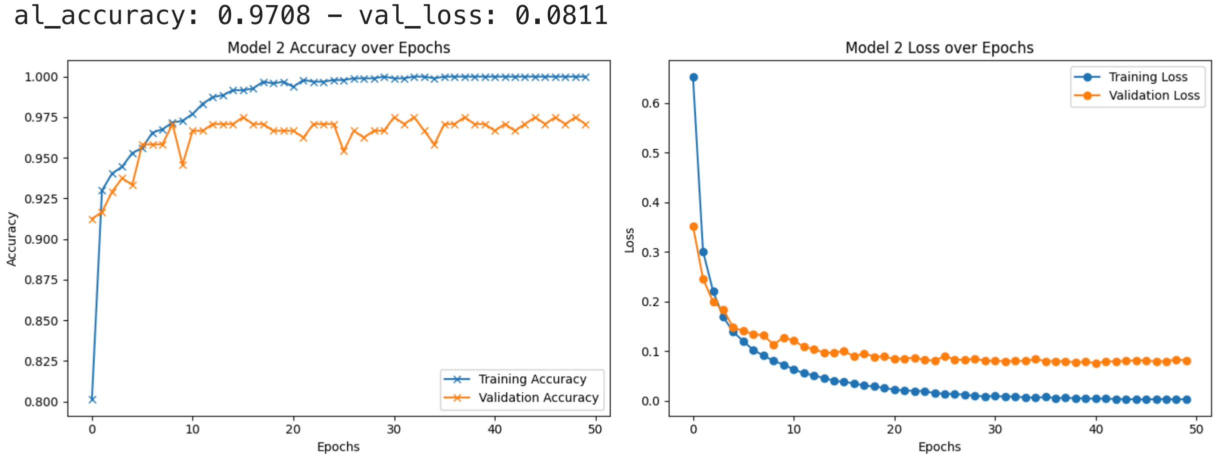
```
Epoch 1/50
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6564 - loss: 0.8440 - v
al_accuracy: 0.9125 - val_loss: 0.3521
Epoch 2/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9094 - loss: 0.3618 - v
al_accuracy: 0.9167 - val_loss: 0.2454
Epoch 3/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9425 - loss: 0.2346 - v
al_accuracy: 0.9292 - val_loss: 0.1988
Epoch 4/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9400 - loss: 0.1752 - v
al_accuracy: 0.9375 - val_loss: 0.1830
Epoch 5/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9563 - loss: 0.1463 - v
al_accuracy: 0.9333 - val_loss: 0.1478
Epoch 6/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9560 - loss: 0.1246 - v
al_accuracy: 0.9583 - val_loss: 0.1404
Epoch 7/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9668 - loss: 0.0937 - v
al_accuracy: 0.9583 - val_loss: 0.1342
Epoch 8/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9722 - loss: 0.0863 - v
al_accuracy: 0.9583 - val_loss: 0.1318
Epoch 9/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9749 - loss: 0.0795 - v
al_accuracy: 0.9708 - val_loss: 0.1129
Epoch 10/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9660 - loss: 0.0813 - v
al_accuracy: 0.9458 - val_loss: 0.1276
Epoch 11/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9824 - loss: 0.0580 - v
al_accuracy: 0.9667 - val_loss: 0.1211
Epoch 12/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9806 - loss: 0.0607 - v
al_accuracy: 0.9667 - val_loss: 0.1087
Epoch 13/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9886 - loss: 0.0530 - v
al_accuracy: 0.9708 - val_loss: 0.1043
Epoch 14/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9901 - loss: 0.0458 - v
al_accuracy: 0.9708 - val_loss: 0.0967
Epoch 15/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9904 - loss: 0.0395 - v
al_accuracy: 0.9708 - val_loss: 0.0966
Epoch 16/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9897 - loss: 0.0410 - v
al_accuracy: 0.9750 - val_loss: 0.1004
Epoch 17/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9897 - loss: 0.0348 - v
al_accuracy: 0.9708 - val_loss: 0.0887
Epoch 18/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9962 - loss: 0.0306 - v
al_accuracy: 0.9708 - val_loss: 0.0956
Epoch 19/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9947 - loss: 0.0280 - v
al_accuracy: 0.9667 - val_loss: 0.0885
```

```
Epoch 20/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9948 - loss: 0.0286 - v
al_accuracy: 0.9667 - val_loss: 0.0894
Epoch 21/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9945 - loss: 0.0215 - v
al_accuracy: 0.9667 - val_loss: 0.0837
Epoch 22/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9982 - loss: 0.0199 - v
al_accuracy: 0.9625 - val_loss: 0.0851
Epoch 23/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9981 - loss: 0.0176 - v
al_accuracy: 0.9708 - val_loss: 0.0867
Epoch 24/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9957 - loss: 0.0210 - v
al_accuracy: 0.9708 - val_loss: 0.0827
Epoch 25/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9961 - loss: 0.0162 - v
al_accuracy: 0.9708 - val_loss: 0.0805
Epoch 26/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9967 - loss: 0.0155 - v
al_accuracy: 0.9542 - val_loss: 0.0901
Epoch 27/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9984 - loss: 0.0123 - v
al_accuracy: 0.9667 - val_loss: 0.0829
Epoch 28/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9998 - loss: 0.0136 - v
al_accuracy: 0.9625 - val_loss: 0.0822
Epoch 29/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9989 - loss: 0.0090 - v
al_accuracy: 0.9667 - val_loss: 0.0845
Epoch 30/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0084 - v
al_accuracy: 0.9667 - val_loss: 0.0805
Epoch 31/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9999 - loss: 0.0079 - v
al_accuracy: 0.9750 - val_loss: 0.0805
Epoch 32/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9985 - loss: 0.0082 - v
al_accuracy: 0.9708 - val_loss: 0.0794
Epoch 33/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 0.0069 - v
al_accuracy: 0.9750 - val_loss: 0.0804
Epoch 34/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0060 - v
al_accuracy: 0.9667 - val_loss: 0.0799
Epoch 35/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9993 - loss: 0.0065 - v
al_accuracy: 0.9583 - val_loss: 0.0838
Epoch 36/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 0.0088 - v
al_accuracy: 0.9708 - val_loss: 0.0799
Epoch 37/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 1.0000 - loss: 0.0045 - v
al_accuracy: 0.9708 - val_loss: 0.0798
Epoch 38/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 1.0000 - loss: 0.0066 -
```

```
val_accuracy: 0.9750 - val_loss: 0.0796
Epoch 39/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 0.0054 - v
al_accuracy: 0.9708 - val_loss: 0.0771
Epoch 40/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 1.0000 - loss: 0.0044 -
val_accuracy: 0.9708 - val_loss: 0.0783
Epoch 41/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 1.0000 - loss: 0.0034 -
val_accuracy: 0.9667 - val_loss: 0.0757
Epoch 42/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 1.0000 - loss: 0.0047 - v
al_accuracy: 0.9708 - val_loss: 0.0793
Epoch 43/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 0.0046 - v
al_accuracy: 0.9667 - val_loss: 0.0792
Epoch 44/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 0.0022 - v
al_accuracy: 0.9708 - val_loss: 0.0801
Epoch 45/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 0.0025 - v
al_accuracy: 0.9750 - val_loss: 0.0816
Epoch 46/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 0.0017 - v
al_accuracy: 0.9708 - val_loss: 0.0811
Epoch 47/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 1.0000 - loss: 0.0046 - v
al_accuracy: 0.9750 - val_loss: 0.0789
Epoch 48/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 0.0034 - v
al_accuracy: 0.9708 - val_loss: 0.0798
Epoch 49/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0021 - v
al_accuracy: 0.9750 - val_loss: 0.0830
Epoch 50/50
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0026 - v
al_accuracy: 0.9708 - val_loss: 0.0811
```




```
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9544 - loss: 0.1629
Loss: 0.17335011065006256
Accuracy: 0.9523411393165588
```

Model 3:

- Changes:
  - Dataset Data Engineering
  - Model Definition
  - Model Compile
  - Model Training

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

def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)


label_encoder = LabelEncoder()
data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])
data

data = data.drop(columns=['StudentID', 'Ethnicity', 'Gender'])

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model3 = Sequential()
model3.add(Dense(64, input_dim=12, activation='relu'))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(3, activation='softmax'))
model3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

history = model3.fit(X_train, y_train, epochs=50, batch_size=10, validation_
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model 3 Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model 3 Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

loss3, accuracy3 = model3.evaluate(X_test, y_test)
print(f"Loss: {loss3}")
print(f"Accuracy: {accuracy3}")
```

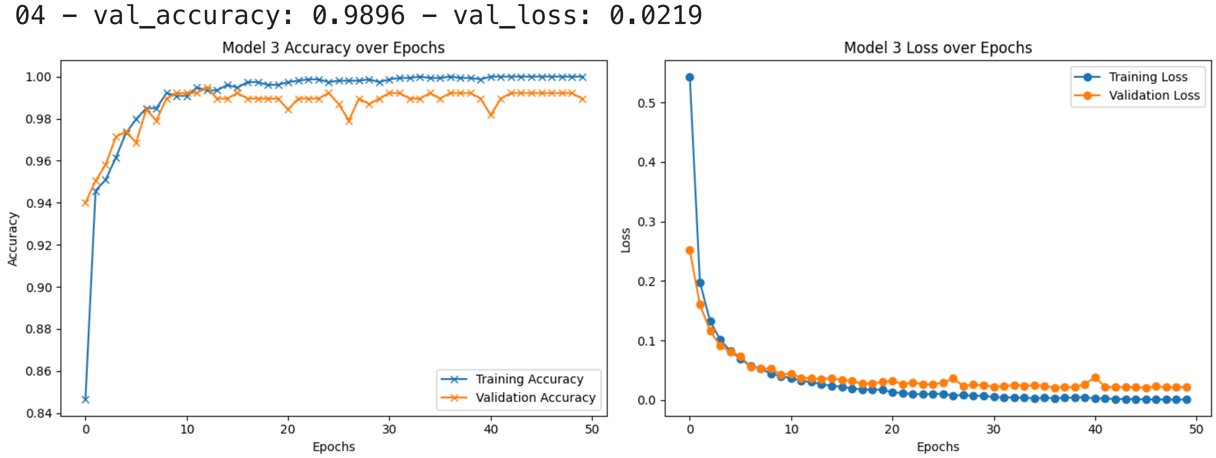
```
Epoch 1/50
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6709 - loss: 0.8104 -
val_accuracy: 0.9399 - val_loss: 0.2515
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9422 - loss: 0.2182 -
val_accuracy: 0.9504 - val_loss: 0.1603
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9401 - loss: 0.1513 -
val_accuracy: 0.9582 - val_loss: 0.1161
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9663 - loss: 0.0871 -
val_accuracy: 0.9713 - val_loss: 0.0915
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9769 - loss: 0.0805 -
val_accuracy: 0.9739 - val_loss: 0.0814
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9778 - loss: 0.0738 -
val_accuracy: 0.9687 - val_loss: 0.0732
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9833 - loss: 0.0534 -
val_accuracy: 0.9843 - val_loss: 0.0563
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9861 - loss: 0.0528 -
val_accuracy: 0.9791 - val_loss: 0.0531
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9940 - loss: 0.0422 -
val_accuracy: 0.9896 - val_loss: 0.0528
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9891 - loss: 0.0445 -
val_accuracy: 0.9922 - val_loss: 0.0426
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9908 - loss: 0.0339 -
val_accuracy: 0.9922 - val_loss: 0.0445
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9937 - loss: 0.0347 -
val_accuracy: 0.9922 - val_loss: 0.0371
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9949 - loss: 0.0308 -
val_accuracy: 0.9948 - val_loss: 0.0362
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9933 - loss: 0.0293 -
val_accuracy: 0.9896 - val_loss: 0.0351
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9986 - loss: 0.0186 -
val_accuracy: 0.9896 - val_loss: 0.0370
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9958 - loss: 0.0216 -
val_accuracy: 0.9922 - val_loss: 0.0331
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9992 - loss: 0.0156 -
val_accuracy: 0.9896 - val_loss: 0.0324
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9985 - loss: 0.0129 -
val_accuracy: 0.9896 - val_loss: 0.0273
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9958 - loss: 0.0182 -
val_accuracy: 0.9896 - val_loss: 0.0278
```

```
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9947 - loss: 0.0197 -
val_accuracy: 0.9896 - val_loss: 0.0302
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9990 - loss: 0.0087 -
val_accuracy: 0.9843 - val_loss: 0.0326
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9991 - loss: 0.0080 -
val_accuracy: 0.9896 - val_loss: 0.0263
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9996 - loss: 0.0099 -
val_accuracy: 0.9896 - val_loss: 0.0291
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9989 - loss: 0.0087 -
val_accuracy: 0.9896 - val_loss: 0.0264
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9963 - loss: 0.0111 -
val_accuracy: 0.9922 - val_loss: 0.0261
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9983 - loss: 0.0094 -
val_accuracy: 0.9869 - val_loss: 0.0291
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9993 - loss: 0.0056 -
val_accuracy: 0.9791 - val_loss: 0.0364
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9953 - loss: 0.0112 -
val_accuracy: 0.9896 - val_loss: 0.0233
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9988 - loss: 0.0070 -
val_accuracy: 0.9869 - val_loss: 0.0259
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9975 - loss: 0.0067 -
val_accuracy: 0.9896 - val_loss: 0.0249
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9992 - loss: 0.0045 -
val_accuracy: 0.9922 - val_loss: 0.0222
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9993 - loss: 0.0043 -
val_accuracy: 0.9922 - val_loss: 0.0230
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0038 -
val_accuracy: 0.9896 - val_loss: 0.0255
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0041 -
val_accuracy: 0.9896 - val_loss: 0.0237
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9998 - loss: 0.0024 -
val_accuracy: 0.9922 - val_loss: 0.0247
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0031 -
val_accuracy: 0.9896 - val_loss: 0.0237
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0037 -
val_accuracy: 0.9922 - val_loss: 0.0205
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9999 - loss: 0.0017 -
```

```
val_accuracy: 0.9922 - val_loss: 0.0220
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0048 -
val_accuracy: 0.9922 - val_loss: 0.0212
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9996 - loss: 0.0035 -
val_accuracy: 0.9896 - val_loss: 0.0269
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0018 -
val_accuracy: 0.9817 - val_loss: 0.0378
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0029 -
val_accuracy: 0.9896 - val_loss: 0.0225
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 1.0000 - loss: 0.0012 -
val_accuracy: 0.9922 - val_loss: 0.0212
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0011 -
val_accuracy: 0.9922 - val_loss: 0.0222
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0016 -
val_accuracy: 0.9922 - val_loss: 0.0216
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 9.3887e-
04 - val_accuracy: 0.9922 - val_loss: 0.0208
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 7.2404e-
04 - val_accuracy: 0.9922 - val_loss: 0.0230
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 7.0624e-
04 - val_accuracy: 0.9922 - val_loss: 0.0217
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 6.2985e-
04 - val_accuracy: 0.9922 - val_loss: 0.0218
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 7.0480e-
04 - val_accuracy: 0.9896 - val_loss: 0.0219
```




```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9861 - loss: 0.0677
Loss: 0.082427978515625
Accuracy: 0.9812108278274536
```

In [ ]:
```
model_data = [
    {"Model": "Model 1", "Loss": loss, "Accuracy": accuracy},
```

```
    {"Model": "Model 2", "Loss": loss2, "Accuracy": accuracy2},
    {"Model": "Model 3", "Loss": loss3, "Accuracy": accuracy3}
]


df_results = pd.DataFrame(model_data)

print(df_results)
```

```
     Model      Loss  Accuracy
0  Model 1  0.067891  0.957358
1  Model 2  0.173350  0.952341
2  Model 3  0.082428  0.981211
```

In [ ]:

```
num_students = 5
random_indices = np.random.choice(X_test.shape[0], size=num_students, replac
X_sample = X_test[random_indices]
y_sample = y_test.iloc[random_indices].values

print(f"X_sample shape: {X_sample.shape}")

y_pred_model1 = model.predict(X_sample)
y_pred_model1_labels = np.argmax(y_pred_model1, axis=1)

y_pred_model2 = model2.predict(X_sample[:, :11])
y_pred_model2_labels = np.argmax(y_pred_model2, axis=1)

y_pred_model3 = model3.predict(X_sample)
y_pred_model3_labels = np.argmax(y_pred_model3, axis=1)

profile_mapping = {0: 'Low', 1: 'Medium', 2: 'High'}
results = []
for i in range(num_students):
    actual_profile = profile_mapping[y_sample[i]]
    model1_pred_profile = profile_mapping[y_pred_model1_labels[i]]
    model2_pred_profile = profile_mapping[y_pred_model2_labels[i]]
    model3_pred_profile = profile_mapping[y_pred_model3_labels[i]]

    results.append([f"Student {i+1}", actual_profile, model1_pred_profile, m

df_predictions = pd.DataFrame(results, columns=["Student", "Actual Profile",

print(df_predictions)
```

```
X_sample shape: (5, 12)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
     Student Actual Profile Model 1 Model 2 Model 3
0  Student 1         Medium  Medium  Medium  Medium
1  Student 2         Medium  Medium  Medium  Medium
2  Student 3           High    High    High    High
3  Student 4         Medium  Medium  Medium  Medium
4  Student 5           High    High    High    High
```

Cual fue el mejor de los tres modelos anteriores?

Se pudo determinar que el mejor modeloo de los tres fue el tercero, ya que ahi se obtuvo una de las mejores graficas y un accurracy de 0.957358

Use the Student GPA dataset to predict student GPA.

Use previous concepts to create different Neural Network Architectures and compare your results. (Python Notebook)

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.

Create a comparative table and upload you code and the comparative table as the activity evidence.

Experiment 1: A single Dense Hidden Layer

In [5]:
```
data = pd.read_csv("Student_performance_data _.csv")

def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)


label_encoder = LabelEncoder()
data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])
data

data = data.drop(columns=['StudentID', 'Ethnicity', 'Gender'])

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model4 = Sequential()
model4.add(Dense(64, input_dim=12, activation='relu'))
model4.add(Dense(32, activation='relu'))
model4.add(Dense(3, activation='softmax'))
model4.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

history = model4.fit(X_train, y_train, epochs=50, batch_size=10, validation_
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model 4 Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model 4 Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

loss4, accuracy4 = model4.evaluate(X_test, y_test)
print(f"Loss: {loss4}")
print(f"Accuracy: {accuracy4}")
```

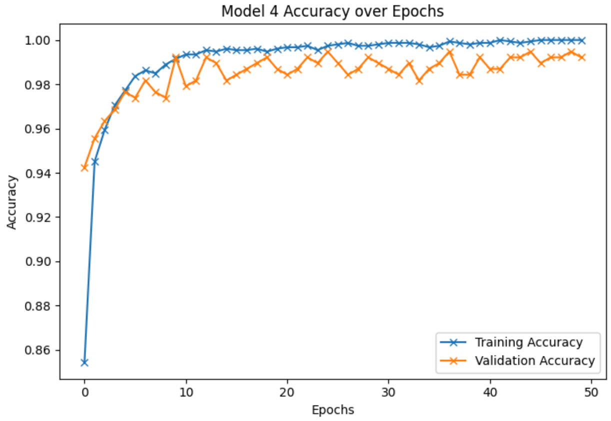
```
Epoch 1/50
```

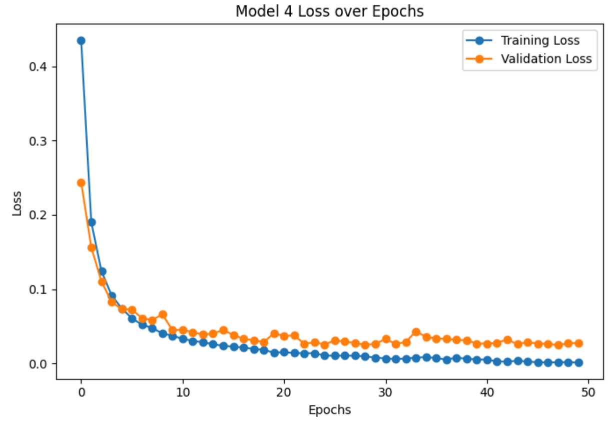
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7421 - loss: 0.6316 -
val_accuracy: 0.9426 - val_loss: 0.2442
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9340 - loss: 0.2281 -
val_accuracy: 0.9556 - val_loss: 0.1561
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9581 - loss: 0.1343 -
val_accuracy: 0.9634 - val_loss: 0.1105
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9741 - loss: 0.0825 -
val_accuracy: 0.9687 - val_loss: 0.0826
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9799 - loss: 0.0627 -
val_accuracy: 0.9765 - val_loss: 0.0730
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9870 - loss: 0.0584 -
val_accuracy: 0.9739 - val_loss: 0.0724
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9880 - loss: 0.0450 -
val_accuracy: 0.9817 - val_loss: 0.0608
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9886 - loss: 0.0383 -
val_accuracy: 0.9765 - val_loss: 0.0581
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9908 - loss: 0.0338 -
val_accuracy: 0.9739 - val_loss: 0.0661
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9951 - loss: 0.0301 -
val_accuracy: 0.9922 - val_loss: 0.0447
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9961 - loss: 0.0290 -
val_accuracy: 0.9791 - val_loss: 0.0453
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9940 - loss: 0.0231 -
val_accuracy: 0.9817 - val_loss: 0.0419
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9941 - loss: 0.0242 -
val_accuracy: 0.9922 - val_loss: 0.0386
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9955 - loss: 0.0211 -
val_accuracy: 0.9896 - val_loss: 0.0403
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9987 - loss: 0.0204 -
val_accuracy: 0.9817 - val_loss: 0.0447
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9938 - loss: 0.0245 -
val_accuracy: 0.9843 - val_loss: 0.0382
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9984 - loss: 0.0133 -
val_accuracy: 0.9869 - val_loss: 0.0329
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9977 - loss: 0.0217 -
val_accuracy: 0.9896 - val_loss: 0.0312
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9955 - loss: 0.0151 -
val_accuracy: 0.9922 - val_loss: 0.0284
```

```
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9971 - loss: 0.0112 -
val_accuracy: 0.9869 - val_loss: 0.0405
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9972 - loss: 0.0139 -
val_accuracy: 0.9843 - val_loss: 0.0370
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9973 - loss: 0.0138 -
val_accuracy: 0.9869 - val_loss: 0.0374
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9968 - loss: 0.0128 -
val_accuracy: 0.9922 - val_loss: 0.0262
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9969 - loss: 0.0113 -
val_accuracy: 0.9896 - val_loss: 0.0283
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9972 - loss: 0.0108 -
val_accuracy: 0.9948 - val_loss: 0.0253
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9962 - loss: 0.0137 -
val_accuracy: 0.9896 - val_loss: 0.0309
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9995 - loss: 0.0069 -
val_accuracy: 0.9843 - val_loss: 0.0293
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9987 - loss: 0.0067 -
val_accuracy: 0.9869 - val_loss: 0.0271
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9959 - loss: 0.0120 -
val_accuracy: 0.9922 - val_loss: 0.0250
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9985 - loss: 0.0070 -
val_accuracy: 0.9896 - val_loss: 0.0260
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9993 - loss: 0.0054 -
val_accuracy: 0.9869 - val_loss: 0.0326
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9995 - loss: 0.0037 -
val_accuracy: 0.9843 - val_loss: 0.0262
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9998 - loss: 0.0044 -
val_accuracy: 0.9896 - val_loss: 0.0284
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9995 - loss: 0.0051 -
val_accuracy: 0.9817 - val_loss: 0.0432
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9968 - loss: 0.0074 -
val_accuracy: 0.9869 - val_loss: 0.0355
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9995 - loss: 0.0042 -
val_accuracy: 0.9896 - val_loss: 0.0336
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9984 - loss: 0.0070 -
val_accuracy: 0.9948 - val_loss: 0.0328
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9995 - loss: 0.0043 -
```

```
val_accuracy: 0.9843 - val_loss: 0.0315
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9976 - loss: 0.0059 -
val_accuracy: 0.9843 - val_loss: 0.0310
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9997 - loss: 0.0032 -
val_accuracy: 0.9922 - val_loss: 0.0266
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9962 - loss: 0.0109 -
val_accuracy: 0.9869 - val_loss: 0.0266
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0029 -
val_accuracy: 0.9869 - val_loss: 0.0276
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9999 - loss: 0.0015 -
val_accuracy: 0.9922 - val_loss: 0.0322
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9994 - loss: 0.0024 -
val_accuracy: 0.9922 - val_loss: 0.0256
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9998 - loss: 0.0017 -
val_accuracy: 0.9948 - val_loss: 0.0284
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0019 -
val_accuracy: 0.9896 - val_loss: 0.0265
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0012 -
val_accuracy: 0.9922 - val_loss: 0.0262
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 9.2851e-
04 - val_accuracy: 0.9922 - val_loss: 0.0245
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0019 -
val_accuracy: 0.9948 - val_loss: 0.0268
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 7.0665e-
04 - val_accuracy: 0.9922 - val_loss: 0.0273
```






```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9841 - loss: 0.0719
Loss: 0.10427138209342957
Accuracy: 0.9728600978851318
```

Experiment 2: A set of three Dense Hidden Layers

In [6]:
```
data = pd.read_csv("Student_performance_data _.csv")

def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)


label_encoder = LabelEncoder()
data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])
data

data = data.drop(columns=['StudentID', 'Ethnicity', 'Gender'])

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model5 = Sequential()
model5.add(Dense(64, input_dim=12, activation='relu'))
model5.add(Dense(32, activation='relu'))
model5.add(Dense(32, activation='relu'))
model5.add(Dense(32, activation='relu'))
model5.add(Dense(3, activation='softmax'))
model5.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

history = model5.fit(X_train, y_train, epochs=50, batch_size=10, validation_
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model 5 Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
```

```
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model 5 Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

loss5, accuracy5 = model5.evaluate(X_test, y_test)
print(f"Loss: {loss5}")
print(f"Accuracy: {accuracy5}")
```

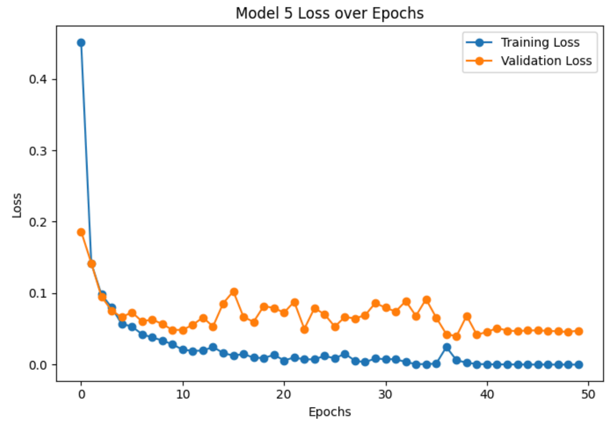
```
Epoch 1/50
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.7180 - loss: 0.6835 -
val_accuracy: 0.9399 - val_loss: 0.1864
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9412 - loss: 0.1736 -
val_accuracy: 0.9373 - val_loss: 0.1422
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9698 - loss: 0.0977 -
val_accuracy: 0.9634 - val_loss: 0.0954
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9706 - loss: 0.0742 -
val_accuracy: 0.9713 - val_loss: 0.0759
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9826 - loss: 0.0595 -
val_accuracy: 0.9739 - val_loss: 0.0663
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9826 - loss: 0.0523 -
val_accuracy: 0.9713 - val_loss: 0.0734
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9828 - loss: 0.0391 -
val_accuracy: 0.9817 - val_loss: 0.0606
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9864 - loss: 0.0402 -
val_accuracy: 0.9739 - val_loss: 0.0632
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9923 - loss: 0.0316 -
val_accuracy: 0.9791 - val_loss: 0.0569
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9883 - loss: 0.0382 -
val_accuracy: 0.9843 - val_loss: 0.0486
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9932 - loss: 0.0220 -
val_accuracy: 0.9817 - val_loss: 0.0487
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9953 - loss: 0.0159 -
val_accuracy: 0.9791 - val_loss: 0.0557
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9956 - loss: 0.0143 -
val_accuracy: 0.9791 - val_loss: 0.0660
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9901 - loss: 0.0247 -
val_accuracy: 0.9791 - val_loss: 0.0535
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9981 - loss: 0.0136 -
val_accuracy: 0.9739 - val_loss: 0.0858
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9952 - loss: 0.0100 -
val_accuracy: 0.9608 - val_loss: 0.1026
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9959 - loss: 0.0167 -
val_accuracy: 0.9791 - val_loss: 0.0670
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9976 - loss: 0.0077 -
val_accuracy: 0.9817 - val_loss: 0.0598
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9994 - loss: 0.0069 -
val_accuracy: 0.9843 - val_loss: 0.0821
```

```
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9979 - loss: 0.0087 -
val_accuracy: 0.9739 - val_loss: 0.0793
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9994 - loss: 0.0042 -
val_accuracy: 0.9713 - val_loss: 0.0729
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9961 - loss: 0.0080 -
val_accuracy: 0.9713 - val_loss: 0.0881
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9965 - loss: 0.0092 -
val_accuracy: 0.9791 - val_loss: 0.0501
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9998 - loss: 0.0018 -
val_accuracy: 0.9687 - val_loss: 0.0792
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9978 - loss: 0.0086 -
val_accuracy: 0.9687 - val_loss: 0.0704
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9982 - loss: 0.0077 -
val_accuracy: 0.9896 - val_loss: 0.0533
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9942 - loss: 0.0127 -
val_accuracy: 0.9739 - val_loss: 0.0667
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9990 - loss: 0.0055 -
val_accuracy: 0.9869 - val_loss: 0.0647
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9993 - loss: 0.0046 -
val_accuracy: 0.9869 - val_loss: 0.0694
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9956 - loss: 0.0097 -
val_accuracy: 0.9843 - val_loss: 0.0863
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9970 - loss: 0.0056 -
val_accuracy: 0.9765 - val_loss: 0.0797
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9982 - loss: 0.0035 -
val_accuracy: 0.9817 - val_loss: 0.0744
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9996 - loss: 0.0028 -
val_accuracy: 0.9765 - val_loss: 0.0890
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0011 -
val_accuracy: 0.9817 - val_loss: 0.0683
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 5.0832e-
04 - val_accuracy: 0.9765 - val_loss: 0.0917
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9999 - loss: 0.0011 -
val_accuracy: 0.9765 - val_loss: 0.0659
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9953 - loss: 0.0190 -
val_accuracy: 0.9869 - val_loss: 0.0428
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9997 - loss: 0.0023 -
```

```
val_accuracy: 0.9869 - val_loss: 0.0399
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9979 - loss: 0.0058 -
val_accuracy: 0.9713 - val_loss: 0.0684
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0012 -
val_accuracy: 0.9869 - val_loss: 0.0420
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 4.9560e-
04 - val_accuracy: 0.9869 - val_loss: 0.0457
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 2.7137e-
04 - val_accuracy: 0.9765 - val_loss: 0.0513
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 3.0426e-
04 - val_accuracy: 0.9817 - val_loss: 0.0477
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 2.1138e-
04 - val_accuracy: 0.9843 - val_loss: 0.0470
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 1.6297e-
04 - val_accuracy: 0.9817 - val_loss: 0.0479
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 1.3911e-
04 - val_accuracy: 0.9817 - val_loss: 0.0479
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 1.0222e-
04 - val_accuracy: 0.9843 - val_loss: 0.0473
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 9.4618e-
05 - val_accuracy: 0.9869 - val_loss: 0.0467
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 1.2866e-
04 - val_accuracy: 0.9869 - val_loss: 0.0458
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 9.1051e-
05 - val_accuracy: 0.9869 - val_loss: 0.0477
```




```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9889 - loss: 0.0760
Loss: 0.09529072046279907
Accuracy: 0.9832985401153564
```

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

In [7]:
```
data = pd.read_csv("Student_performance_data _.csv")

def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)


label_encoder = LabelEncoder()
data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])
data

data = data.drop(columns=['StudentID', 'Ethnicity', 'Gender'])

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model6 = Sequential()
model6.add(Dense(64, input_dim=12, activation='relu'))
model6.add(Dense(32, activation='relu'))
model6.add(Dropout(0.5))
model6.add(Dense(32, activation='relu'))
model6.add(Dropout(0.5))
model6.add(Dense(32, activation='relu'))
model6.add(Dropout(0.5))
model6.add(Dense(3, activation='softmax'))
model6.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

history = model6.fit(X_train, y_train, epochs=50, batch_size=10, validation_
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model 6 Accuracy over Epochs')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model 6 Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

loss6, accuracy6 = model6.evaluate(X_test, y_test)
print(f"Loss: {loss6}")
print(f"Accuracy: {accuracy6}")
```

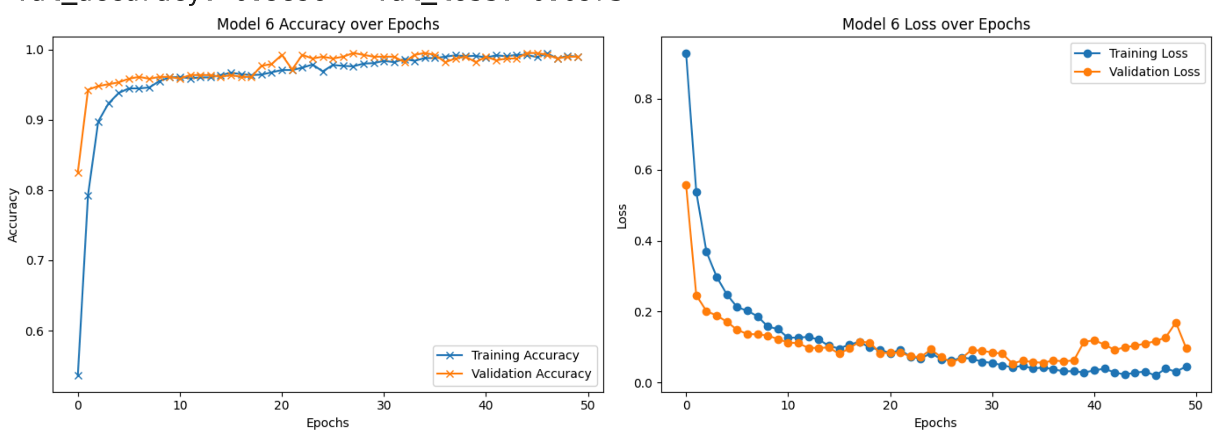
```
Epoch 1/50
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4269 - loss: 1.0691 -
val_accuracy: 0.8251 - val_loss: 0.5576
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7316 - loss: 0.6380 -
val_accuracy: 0.9426 - val_loss: 0.2460
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8887 - loss: 0.3887 -
val_accuracy: 0.9478 - val_loss: 0.2015
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9194 - loss: 0.3075 -
val_accuracy: 0.9504 - val_loss: 0.1890
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9437 - loss: 0.2354 -
val_accuracy: 0.9530 - val_loss: 0.1714
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9463 - loss: 0.2105 -
val_accuracy: 0.9582 - val_loss: 0.1487
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9484 - loss: 0.1959 -
val_accuracy: 0.9608 - val_loss: 0.1369
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9449 - loss: 0.1893 -
val_accuracy: 0.9582 - val_loss: 0.1358
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9524 - loss: 0.1714 -
val_accuracy: 0.9608 - val_loss: 0.1327
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9572 - loss: 0.1675 -
val_accuracy: 0.9608 - val_loss: 0.1225
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9636 - loss: 0.1315 -
val_accuracy: 0.9582 - val_loss: 0.1131
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9585 - loss: 0.1379 -
val_accuracy: 0.9634 - val_loss: 0.1108
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9623 - loss: 0.1282 -
val_accuracy: 0.9634 - val_loss: 0.0976
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9678 - loss: 0.1151 -
val_accuracy: 0.9634 - val_loss: 0.0978
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9673 - loss: 0.0974 -
val_accuracy: 0.9608 - val_loss: 0.0987
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9605 - loss: 0.0961 -
val_accuracy: 0.9634 - val_loss: 0.0811
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9666 - loss: 0.1095 -
val_accuracy: 0.9608 - val_loss: 0.0963
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9605 - loss: 0.1270 -
val_accuracy: 0.9608 - val_loss: 0.1155
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9708 - loss: 0.0864 -
val_accuracy: 0.9765 - val_loss: 0.1129
```

```
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9683 - loss: 0.0910 -
val_accuracy: 0.9791 - val_loss: 0.0834
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9769 - loss: 0.0640 -
val_accuracy: 0.9922 - val_loss: 0.0837
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9742 - loss: 0.0885 -
val_accuracy: 0.9713 - val_loss: 0.0846
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9696 - loss: 0.0768 -
val_accuracy: 0.9922 - val_loss: 0.0736
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9768 - loss: 0.0651 -
val_accuracy: 0.9869 - val_loss: 0.0720
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9709 - loss: 0.0713 -
val_accuracy: 0.9896 - val_loss: 0.0934
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9764 - loss: 0.0645 -
val_accuracy: 0.9869 - val_loss: 0.0716
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9738 - loss: 0.0613 -
val_accuracy: 0.9896 - val_loss: 0.0576
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9776 - loss: 0.0610 -
val_accuracy: 0.9948 - val_loss: 0.0682
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9841 - loss: 0.0572 -
val_accuracy: 0.9922 - val_loss: 0.0927
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9764 - loss: 0.0637 -
val_accuracy: 0.9896 - val_loss: 0.0892
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9848 - loss: 0.0485 -
val_accuracy: 0.9896 - val_loss: 0.0852
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9782 - loss: 0.0645 -
val_accuracy: 0.9896 - val_loss: 0.0811
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9864 - loss: 0.0409 -
val_accuracy: 0.9817 - val_loss: 0.0533
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9772 - loss: 0.0507 -
val_accuracy: 0.9922 - val_loss: 0.0622
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9914 - loss: 0.0267 -
val_accuracy: 0.9948 - val_loss: 0.0572
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9777 - loss: 0.0600 -
val_accuracy: 0.9922 - val_loss: 0.0560
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9865 - loss: 0.0392 -
val_accuracy: 0.9817 - val_loss: 0.0622
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9917 - loss: 0.0319 -
```

```
val_accuracy: 0.9869 - val_loss: 0.0600
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9889 - loss: 0.0448 -
val_accuracy: 0.9896 - val_loss: 0.0624
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9915 - loss: 0.0319 -
val_accuracy: 0.9817 - val_loss: 0.1147
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9897 - loss: 0.0337 -
val_accuracy: 0.9896 - val_loss: 0.1192
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9939 - loss: 0.0315 -
val_accuracy: 0.9843 - val_loss: 0.1072
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9881 - loss: 0.0309 -
val_accuracy: 0.9869 - val_loss: 0.0919
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9908 - loss: 0.0291 -
val_accuracy: 0.9869 - val_loss: 0.0995
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9906 - loss: 0.0291 -
val_accuracy: 0.9948 - val_loss: 0.1036
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9903 - loss: 0.0274 -
val_accuracy: 0.9948 - val_loss: 0.1103
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9945 - loss: 0.0184 -
val_accuracy: 0.9922 - val_loss: 0.1163
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9847 - loss: 0.0446 -
val_accuracy: 0.9869 - val_loss: 0.1275
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9949 - loss: 0.0146 -
val_accuracy: 0.9896 - val_loss: 0.1696
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9877 - loss: 0.0494 -
val_accuracy: 0.9896 - val_loss: 0.0973
```




```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9752 - loss: 0.1041
Loss: 0.16927234828472137
Accuracy: 0.9770354628562927
```

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

In [9]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

data = pd.read_csv("Student_performance_data _.csv")

def assign_profile(gpa):
  if 0 <= gpa <= 2:
    return 'Low'
  elif 2 < gpa <= 3.5:
    return 'Medium'
  elif 3.5 < gpa <= 5:
    return 'High'
  else:
    return 'Unknown'

data['Profile'] = data['GPA'].apply(assign_profile)


label_encoder = LabelEncoder()
data['Profile_Decode'] = label_encoder.fit_transform(data['Profile'])
data

data = data.drop(columns=['StudentID', 'Ethnicity', 'Gender'])

X = data.drop(['Profile', 'Profile_Decode'], axis=1)
y = data['Profile_Decode']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model7 = Sequential()
model7.add(Dense(64, input_dim=12, activation='relu'))
model7.add(Dense(32, activation='relu'))
model7.add(Dropout(0.5))
model7.add(BatchNormalization())
model7.add(Dense(32, activation='relu'))
model7.add(Dropout(0.5))
model7.add(BatchNormalization())

model7.add(Dense(32, activation='relu'))
model7.add(Dropout(0.5))
model7.add(BatchNormalization())

model7.add(Dense(3, activation='softmax'))
model7.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met
```

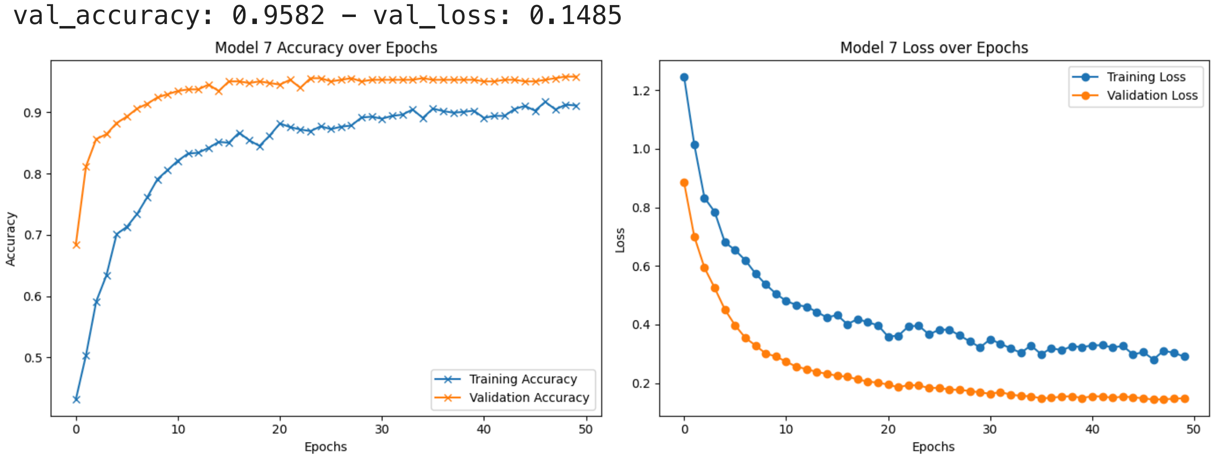
```
history = model7.fit(X_train, y_train, epochs=50, batch_size=10, validation_
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='x')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marke
plt.title('Model 7 Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model 7 Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

loss7, accuracy7 = model7.evaluate(X_test, y_test)
print(f"Loss: {loss7}")
print(f"Accuracy: {accuracy7}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
en using Sequential models, prefer using an `Input(shape)` object as the fir
st layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.4031 - loss: 1.3407 -
val_accuracy: 0.6841 - val_loss: 0.8853
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.4766 - loss: 1.0662
- val_accuracy: 0.8120 - val_loss: 0.7005
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.5873 - loss: 0.8541
- val_accuracy: 0.8564 - val_loss: 0.5938
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.6217 - loss: 0.7901 -
val_accuracy: 0.8642 - val_loss: 0.5247
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7024 - loss: 0.6815 -
val_accuracy: 0.8825 - val_loss: 0.4512
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.6871 - loss: 0.6836 -
val_accuracy: 0.8930 - val_loss: 0.3971
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.7215 - loss: 0.6425 -
val_accuracy: 0.9060 - val_loss: 0.3550
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7683 - loss: 0.5739 -
val_accuracy: 0.9138 - val_loss: 0.3282
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7906 - loss: 0.5259 -
val_accuracy: 0.9243 - val_loss: 0.3012
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.8025 - loss: 0.5274
- val_accuracy: 0.9295 - val_loss: 0.2909
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.8041 - loss: 0.5132
- val_accuracy: 0.9347 - val_loss: 0.2735
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.8430 - loss: 0.4428 -
val_accuracy: 0.9373 - val_loss: 0.2558
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8248 - loss: 0.4993 -
val_accuracy: 0.9373 - val_loss: 0.2481
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8273 - loss: 0.4591 -
val_accuracy: 0.9452 - val_loss: 0.2380
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8486 - loss: 0.4358 -
val_accuracy: 0.9347 - val_loss: 0.2318
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8563 - loss: 0.4229 -
val_accuracy: 0.9504 - val_loss: 0.2251
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8844 - loss: 0.3766 -
val_accuracy: 0.9504 - val_loss: 0.2219
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8403 - loss: 0.4477 -
val_accuracy: 0.9478 - val_loss: 0.2127
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8466 - loss: 0.4100 -
```

```
val_accuracy: 0.9504 - val_loss: 0.2055
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8745 - loss: 0.3669 -
val_accuracy: 0.9478 - val_loss: 0.2015
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8913 - loss: 0.3379 -
val_accuracy: 0.9452 - val_loss: 0.1950
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8887 - loss: 0.3401 -
val_accuracy: 0.9530 - val_loss: 0.1855
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8707 - loss: 0.3947 -
val_accuracy: 0.9399 - val_loss: 0.1937
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8785 - loss: 0.3737 -
val_accuracy: 0.9556 - val_loss: 0.1915
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8544 - loss: 0.4100 -
val_accuracy: 0.9556 - val_loss: 0.1838
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8776 - loss: 0.3666 -
val_accuracy: 0.9504 - val_loss: 0.1833
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8777 - loss: 0.3745 -
val_accuracy: 0.9530 - val_loss: 0.1782
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8660 - loss: 0.3949 -
val_accuracy: 0.9556 - val_loss: 0.1766
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8865 - loss: 0.3557 -
val_accuracy: 0.9504 - val_loss: 0.1733
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8774 - loss: 0.3779 -
val_accuracy: 0.9530 - val_loss: 0.1682
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9068 - loss: 0.3002 -
val_accuracy: 0.9530 - val_loss: 0.1635
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9021 - loss: 0.3235 -
val_accuracy: 0.9530 - val_loss: 0.1692
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8977 - loss: 0.3197 -
val_accuracy: 0.9530 - val_loss: 0.1608
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8966 - loss: 0.3226 -
val_accuracy: 0.9530 - val_loss: 0.1564
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8983 - loss: 0.3146 -
val_accuracy: 0.9556 - val_loss: 0.1551
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9033 - loss: 0.2986 -
val_accuracy: 0.9530 - val_loss: 0.1478
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9077 - loss: 0.3021 -
val_accuracy: 0.9530 - val_loss: 0.1500
Epoch 38/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9021 - loss: 0.2994 -
val_accuracy: 0.9530 - val_loss: 0.1537
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9102 - loss: 0.3054 -
val_accuracy: 0.9530 - val_loss: 0.1553
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9030 - loss: 0.3294 -
val_accuracy: 0.9530 - val_loss: 0.1493
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8855 - loss: 0.3382 -
val_accuracy: 0.9504 - val_loss: 0.1550
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8903 - loss: 0.3329 -
val_accuracy: 0.9504 - val_loss: 0.1544
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9003 - loss: 0.3106 -
val_accuracy: 0.9530 - val_loss: 0.1499
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9020 - loss: 0.3191 -
val_accuracy: 0.9530 - val_loss: 0.1541
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9177 - loss: 0.2868 -
val_accuracy: 0.9504 - val_loss: 0.1516
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9054 - loss: 0.3119 -
val_accuracy: 0.9504 - val_loss: 0.1481
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9070 - loss: 0.2758 -
val_accuracy: 0.9530 - val_loss: 0.1438
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8977 - loss: 0.3268 -
val_accuracy: 0.9556 - val_loss: 0.1450
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9197 - loss: 0.2898 -
val_accuracy: 0.9582 - val_loss: 0.1466
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9114 - loss: 0.2988 -
val_accuracy: 0.9582 - val_loss: 0.1485
```




```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9611 - loss: 0.1493
Loss: 0.14587055146694183
Accuracy: 0.9603340029716492
```

Create a comparative table and upload you code and the comparative table as the activity evidence.

In [16]:
```
model_data2 = [
    {"Model": "Model 4", "Cantidad de Dropouts": "0", "Loss": loss4, "Accura
    {"Model": "Model 5", "Cantidad de Dropouts": "0","Loss": loss5, "Accurac
    {"Model": "Model 6", "Cantidad de Dropouts": "3","Loss": loss6, "Accurac
    {"Model": "Model 7", "Cantidad de Dropouts": "3","Loss": loss7, "Accurac

]


df_results2 = pd.DataFrame(model_data2)

print(df_results2)
```

```
     Model Cantidad de Dropouts      Loss  Accuracy
0  Model 4                    0  0.104271  0.972860
1  Model 5                    0  0.095291  0.983299
2  Model 6                    3  0.169272  0.977035
3  Model 7                    3  0.145871  0.960334
```

Conclusion

En conclusión, se puede ver que el experimento 2 fue el que mejor resultado nos dio, logrando tener el mejor accuracy de 0.983299, el cual logra superar en gran medida a los otros modelos, por lo que podemos decir que el la mejor configuración de red neuronal sería agregar tres sets de capas ocultas, ya que de esa manera, se obtiene una mejora a las capacidades de reconocimiento de patrones al mismo tiempo que se evita el uso de dropouts y batch normalization, esto debido a que los datos son más que suficientes para ser analizados sin la necesidad de su preparación.