Actividad 8 - Dense + Dropout + Batch Normalization

Héctor Hibran Tapia Fernández - A01661114

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 [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
```

2) Load Data

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

In [2]:
```
data = pd.read_csv("Student_performance_data _.csv")
data
```

Out[2]:

	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 [3]:

```
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 [4]:
```
data['Profile'] = pd.cut(data['GPA'], bins=[0, 2, 3.5, 5], labels=['Low', 'M
data.head(10)
```

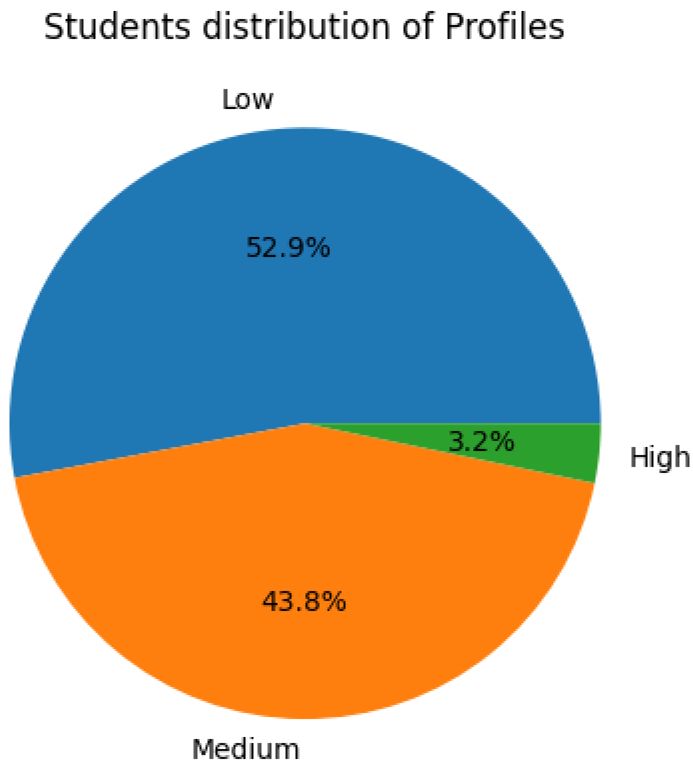
Out[4]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences
0	1001	17	1	0	2	19.833723	7
1	1002	18	0	0	1	15.408756	0
2	1003	15	0	2	3	4.210570	26
3	1004	17	1	0	3	10.028829	14
4	1005	17	1	0	2	4.672495	17
5	1006	18	0	0	1	8.191219	0
6	1007	15	0	1	1	15.601680	10
7	1008	15	1	1	4	15.424496	22
8	1009	17	0	0	0	4.562008	1
9	1010	16	1	0	1	18.444466	0

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 [5]:
```
profile_counts = data['Profile'].value_counts()
plt.pie(profile_counts, labels = profile_counts.index, autopct='%1.1f%%')
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.

De esta forma no porque ordena Medium = 2, Low = 1 y High = 0. Necesitamos ordenarlos en orden para que tengan sentido.

In [6]:
```
# label_encoder = LabelEncoder()
# data['Profile'] = label_encoder.fit_transform(data['Profile'])
# data.head(10)
```

In [7]:
```
data['Profile'].fillna('Low', inplace=True) # Rellenamos NAN's porque tiene.
```

```
<ipython-input-7-cf2e18d9fec9>:1: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inpla
ce method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  data['Profile'].fillna('Low', inplace=True) # Rellenamos NAN's porque tien
e.
```

In [8]:
```
# Cambiamos temporalmente los valores para que LabelEncoder los ordene corre
data['Profile'] = data['Profile'].replace({'High': 'A', 'Medium': 'B', 'Low'

label_encoder = LabelEncoder()
data['Profile'] = label_encoder.fit_transform(data['Profile'])
data['Profile'] = data['Profile'].replace({0: 2, 1: 1, 2: 0}) # Invertimos l

data.head(10)
```

```
<ipython-input-8-0d92fa3a5c47>:2: FutureWarning: The behavior of Series.repl
ace (and DataFrame.replace) with CategoricalDtype is deprecated. In a future
version, replace will only be used for cases that preserve the categories. T
o change the categories, use ser.cat.rename_categories instead.
  data['Profile'] = data['Profile'].replace({'High': 'A', 'Medium': 'B', 'Lo
w': 'C'})
```

Out[8]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences
0	1001	17	1	0	2	19.833723	7
1	1002	18	0	0	1	15.408756	0
2	1003	15	0	2	3	4.210570	26
3	1004	17	1	0	3	10.028829	14
4	1005	17	1	0	2	4.672495	17
5	1006	18	0	0	1	8.191219	0
6	1007	15	0	1	1	15.601680	10
7	1008	15	1	1	4	15.424496	22
8	1009	17	0	0	0	4.562008	1
9	1010	16	1	0	1	18.444466	0

Experiment 1: A single Dense Hidden Layer

In [9]:
```
X = data.drop(columns=['Profile'])
y = data['Profile']

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

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

model = Sequential()
model.add(Dense(64, input_shape = (X_train.shape[1],), activation = 'relu'))
model.add(Dense(3, activation = 'softmax'))  # Para mas de dos clases

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',

history = model.fit(X_train, y_train, epochs = 50, batch_size = 10, validati
```

```
/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
192/192 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.6420 - loss: 0.8284 -
val_accuracy: 0.9269 - val_loss: 0.3188
Epoch 2/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9357 - loss: 0.2914 -
val_accuracy: 0.9374 - val_loss: 0.2204
Epoch 3/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9445 - loss: 0.2097 -
val_accuracy: 0.9478 - val_loss: 0.1745
Epoch 4/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9567 - loss: 0.1537 -
val_accuracy: 0.9520 - val_loss: 0.1467
Epoch 5/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9587 - loss: 0.1240 -
val_accuracy: 0.9541 - val_loss: 0.1247
Epoch 6/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9656 - loss: 0.1097 -
val_accuracy: 0.9562 - val_loss: 0.1125
Epoch 7/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9801 - loss: 0.0779 -
val_accuracy: 0.9603 - val_loss: 0.1016
Epoch 8/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9816 - loss: 0.0728 -
val_accuracy: 0.9687 - val_loss: 0.0922
Epoch 9/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9788 - loss: 0.0743 -
val_accuracy: 0.9645 - val_loss: 0.0895
Epoch 10/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9850 - loss: 0.0564 -
val_accuracy: 0.9770 - val_loss: 0.0814
Epoch 11/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9860 - loss: 0.0556 -
val_accuracy: 0.9812 - val_loss: 0.0784
Epoch 12/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9916 - loss: 0.0396 -
val_accuracy: 0.9833 - val_loss: 0.0715
Epoch 13/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9939 - loss: 0.0411 -
val_accuracy: 0.9833 - val_loss: 0.0689
Epoch 14/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9934 - loss: 0.0343 -
val_accuracy: 0.9833 - val_loss: 0.0697
Epoch 15/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9946 - loss: 0.0342 -
val_accuracy: 0.9791 - val_loss: 0.0656
Epoch 16/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9944 - loss: 0.0282 -
val_accuracy: 0.9791 - val_loss: 0.0677
Epoch 17/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9952 - loss: 0.0287 -
val_accuracy: 0.9854 - val_loss: 0.0627
Epoch 18/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9936 - loss: 0.0275 -
val_accuracy: 0.9833 - val_loss: 0.0686
Epoch 19/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9943 - loss: 0.0269 -
```

```
val_accuracy: 0.9854 - val_loss: 0.0612
Epoch 20/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9974 - loss: 0.0198 -
val_accuracy: 0.9854 - val_loss: 0.0593
Epoch 21/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9949 - loss: 0.0251 -
val_accuracy: 0.9854 - val_loss: 0.0590
Epoch 22/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9971 - loss: 0.0223 -
val_accuracy: 0.9833 - val_loss: 0.0601
Epoch 23/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9970 - loss: 0.0191 -
val_accuracy: 0.9833 - val_loss: 0.0605
Epoch 24/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9975 - loss: 0.0178 -
val_accuracy: 0.9854 - val_loss: 0.0589
Epoch 25/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9992 - loss: 0.0133 -
val_accuracy: 0.9833 - val_loss: 0.0589
Epoch 26/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9975 - loss: 0.0144 -
val_accuracy: 0.9812 - val_loss: 0.0640
Epoch 27/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9989 - loss: 0.0146 -
val_accuracy: 0.9812 - val_loss: 0.0608
Epoch 28/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9987 - loss: 0.0131 -
val_accuracy: 0.9791 - val_loss: 0.0590
Epoch 29/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9979 - loss: 0.0119 -
val_accuracy: 0.9854 - val_loss: 0.0590
Epoch 30/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9998 - loss: 0.0127 -
val_accuracy: 0.9854 - val_loss: 0.0577
Epoch 31/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9995 - loss: 0.0097 -
val_accuracy: 0.9833 - val_loss: 0.0575
Epoch 32/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9956 - loss: 0.0126 -
val_accuracy: 0.9791 - val_loss: 0.0594
Epoch 33/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9996 - loss: 0.0087 -
val_accuracy: 0.9812 - val_loss: 0.0638
Epoch 34/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9995 - loss: 0.0077 -
val_accuracy: 0.9854 - val_loss: 0.0574
Epoch 35/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9993 - loss: 0.0085 -
val_accuracy: 0.9791 - val_loss: 0.0601
Epoch 36/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 0.0063 -
val_accuracy: 0.9791 - val_loss: 0.0609
Epoch 37/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9999 - loss: 0.0082 -
val_accuracy: 0.9833 - val_loss: 0.0573
Epoch 38/50
```

```
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9998 - loss: 0.0053 -
val_accuracy: 0.9875 - val_loss: 0.0544
Epoch 39/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0073 -
val_accuracy: 0.9833 - val_loss: 0.0557
Epoch 40/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0072 -
val_accuracy: 0.9833 - val_loss: 0.0599
Epoch 41/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0053 -
val_accuracy: 0.9833 - val_loss: 0.0573
Epoch 42/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0048 -
val_accuracy: 0.9812 - val_loss: 0.0608
Epoch 43/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0043 -
val_accuracy: 0.9854 - val_loss: 0.0586
Epoch 44/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0042 -
val_accuracy: 0.9812 - val_loss: 0.0606
Epoch 45/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0037 -
val_accuracy: 0.9833 - val_loss: 0.0615
Epoch 46/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0032 -
val_accuracy: 0.9812 - val_loss: 0.0642
Epoch 47/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0035 -
val_accuracy: 0.9854 - val_loss: 0.0585
Epoch 48/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 1.0000 - loss: 0.0035 -
val_accuracy: 0.9854 - val_loss: 0.0557
Epoch 49/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 0.0032 -
val_accuracy: 0.9854 - val_loss: 0.0589
Epoch 50/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 0.0024 -
val_accuracy: 0.9833 - val_loss: 0.0596
```

In [10]:

```python
history_dict = history.history
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
epochs = range(1, len(accuracy) + 1)
```

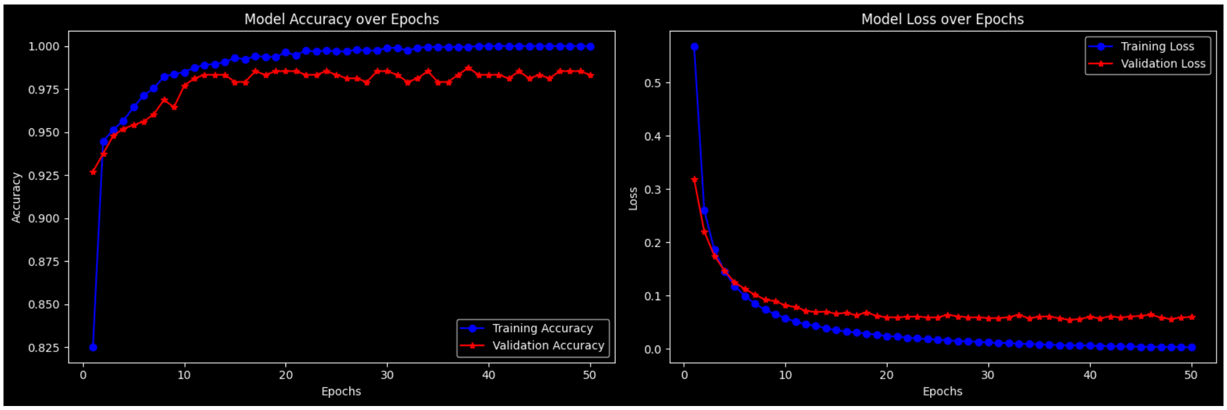
In [11]:

```python
plt.style.use('dark_background')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
ax1.plot(epochs, accuracy, 'bo-', label='Training Accuracy')
ax1.plot(epochs, val_accuracy, 'r*-', label='Validation Accuracy')
ax1.set_title('Model Accuracy over Epochs')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Accuracy')
ax1.legend()
```

```
ax2.plot(epochs, loss, 'bo-', label='Training Loss')
ax2.plot(epochs, val_loss, 'r*-', label='Validation Loss')
ax2.set_title('Model Loss over Epochs')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Loss')
ax2.legend()

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



In [12]:
```
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
label_mapping = {0: 'Low', 1: 'Medium', 2: 'High'}

predicted_labels = [label_mapping[label] for label in predicted_classes]
actual_labels = [label_mapping[label] for label in y_test]

correct_predictions = sum(1 for i in range(len(predicted_labels)) if predict
total_predictions = len(predicted_labels)

# print("Predictions vs Actual Values:")
# for i in range(total_predictions):
#     print(f"Predicted: {predicted_labels[i]}, Actual: {actual_labels[i]}")

print(f"\nTotal correct predictions: {correct_predictions} / {total_predicti
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step

Total correct predictions: 471 / 479
```

In [13]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion

accuracy = accuracy_score(y_test, predicted_classes)
print(f"\nModel Accuracy on Test Data: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, predicted_classes, target_names=['Low',
print("Confusion Matrix:")
print(confusion_matrix(y_test, predicted_classes))
```

```
Model Accuracy on Test Data: 0.98

Classification Report:
              precision    recall  f1-score   support

         Low       1.00      0.99      1.00       249
      Medium       0.97      1.00      0.98       214
        High       0.92      0.69      0.79        16

    accuracy                           0.98       479
   macro avg       0.96      0.89      0.92       479
weighted avg       0.98      0.98      0.98       479

Confusion Matrix:
[[247   2   0]
 [  0 213   1]
 [  0   5  11]]
```

Experiment 2: A set of three Dense Hidden Layers

In [14]:
```python
X = data.drop(columns=['Profile'])
y = data['Profile']

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

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

model = Sequential()
model.add(Dense(128, input_shape=(X_train.shape[1],), activation = 'relu'))
model.add(Dense(64, activation = 'relu'))  # Segunda capa oculta con 64 unic
model.add(Dense(32, activation = 'relu'))  # Tercera capa oculta con 32 unic
model.add(Dense(3, activation = 'softmax')) # Tal vez sean demasiadas capas,

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',

history = model.fit(X_train, y_train, epochs = 50, batch_size = 10, validati
```

```
/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
192/192 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.8339 - loss: 0.4980
- val_accuracy: 0.9415 - val_loss: 0.1475
Epoch 2/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9679 - loss: 0.0942 -
val_accuracy: 0.9729 - val_loss: 0.0925
Epoch 3/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9785 - loss: 0.0750 -
val_accuracy: 0.9562 - val_loss: 0.1046
Epoch 4/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.9829 - loss: 0.0443
- val_accuracy: 0.9770 - val_loss: 0.0689
Epoch 5/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9900 - loss: 0.0432
- val_accuracy: 0.9729 - val_loss: 0.0937
Epoch 6/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9890 - loss: 0.0352 -
val_accuracy: 0.9687 - val_loss: 0.0874
Epoch 7/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9980 - loss: 0.0194 -
val_accuracy: 0.9708 - val_loss: 0.0909
Epoch 8/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9940 - loss: 0.0156 -
val_accuracy: 0.9770 - val_loss: 0.0804
Epoch 9/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9971 - loss: 0.0099 -
val_accuracy: 0.9770 - val_loss: 0.0834
Epoch 10/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9961 - loss: 0.0099 -
val_accuracy: 0.9812 - val_loss: 0.0673
Epoch 11/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9967 - loss: 0.0106 -
val_accuracy: 0.9770 - val_loss: 0.0819
Epoch 12/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9985 - loss: 0.0074 -
val_accuracy: 0.9812 - val_loss: 0.0720
Epoch 13/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9992 - loss: 0.0056 -
val_accuracy: 0.9770 - val_loss: 0.1167
Epoch 14/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9998 - loss: 0.0025 -
val_accuracy: 0.9770 - val_loss: 0.1519
Epoch 15/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9946 - loss: 0.0137 -
val_accuracy: 0.9833 - val_loss: 0.0853
Epoch 16/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9974 - loss: 0.0079 -
val_accuracy: 0.9812 - val_loss: 0.0847
Epoch 17/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 0.0020 -
val_accuracy: 0.9812 - val_loss: 0.0928
Epoch 18/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 8.3332e-
04 - val_accuracy: 0.9812 - val_loss: 0.0945
Epoch 19/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 4.9927e-
```

```
04 - val_accuracy: 0.9812 - val_loss: 0.0943
Epoch 20/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 4.8457e-
04 - val_accuracy: 0.9812 - val_loss: 0.0944
Epoch 21/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 2.8245e-
04 - val_accuracy: 0.9812 - val_loss: 0.0949
Epoch 22/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 2.2672e-
04 - val_accuracy: 0.9812 - val_loss: 0.0957
Epoch 23/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 1.8916e-
04 - val_accuracy: 0.9812 - val_loss: 0.0990
Epoch 24/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 2.0551e-
04 - val_accuracy: 0.9812 - val_loss: 0.0972
Epoch 25/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 1.7862e-
04 - val_accuracy: 0.9812 - val_loss: 0.0985
Epoch 26/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 1.0000 - loss: 1.4255e-
04 - val_accuracy: 0.9812 - val_loss: 0.0992
Epoch 27/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 9.9888e-
05 - val_accuracy: 0.9812 - val_loss: 0.1021
Epoch 28/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 1.2512e-
04 - val_accuracy: 0.9812 - val_loss: 0.1019
Epoch 29/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 9.1631e-
05 - val_accuracy: 0.9812 - val_loss: 0.0991
Epoch 30/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 9.9689e-
05 - val_accuracy: 0.9812 - val_loss: 0.1055
Epoch 31/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 6.6464e-
05 - val_accuracy: 0.9812 - val_loss: 0.1112
Epoch 32/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 7.8280e-
05 - val_accuracy: 0.9812 - val_loss: 0.1067
Epoch 33/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 4.7125e-
05 - val_accuracy: 0.9833 - val_loss: 0.1067
Epoch 34/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 5.4179e-
05 - val_accuracy: 0.9812 - val_loss: 0.1104
Epoch 35/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 3.6633e-
05 - val_accuracy: 0.9812 - val_loss: 0.1048
Epoch 36/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 4.5846e-
05 - val_accuracy: 0.9812 - val_loss: 0.1085
Epoch 37/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 2.4047e-
05 - val_accuracy: 0.9812 - val_loss: 0.1106
Epoch 38/50
```

```
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 3.8609e-05 - val_accuracy: 0.9812 - val_loss: 0.1115
Epoch 39/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 1.0000 - loss: 2.5638e-05 - val_accuracy: 0.9812 - val_loss: 0.1141
Epoch 40/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 1.0000 - loss: 2.7133e-05 - val_accuracy: 0.9812 - val_loss: 0.1117
Epoch 41/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 2.4949e-05 - val_accuracy: 0.9812 - val_loss: 0.1138
Epoch 42/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 1.0000 - loss: 2.3351e-05 - val_accuracy: 0.9812 - val_loss: 0.1121
Epoch 43/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 1.6413e-05 - val_accuracy: 0.9812 - val_loss: 0.1166
Epoch 44/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 1.5387e-05 - val_accuracy: 0.9812 - val_loss: 0.1144
Epoch 45/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 1.7651e-05 - val_accuracy: 0.9812 - val_loss: 0.1183
Epoch 46/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 1.2838e-05 - val_accuracy: 0.9812 - val_loss: 0.1196
Epoch 47/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 1.2934e-05 - val_accuracy: 0.9812 - val_loss: 0.1210
Epoch 48/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 8.4241e-06 - val_accuracy: 0.9812 - val_loss: 0.1200
Epoch 49/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 9.1584e-06 - val_accuracy: 0.9812 - val_loss: 0.1190
Epoch 50/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 1.0000 - loss: 8.8016e-06 - val_accuracy: 0.9812 - val_loss: 0.1163
```

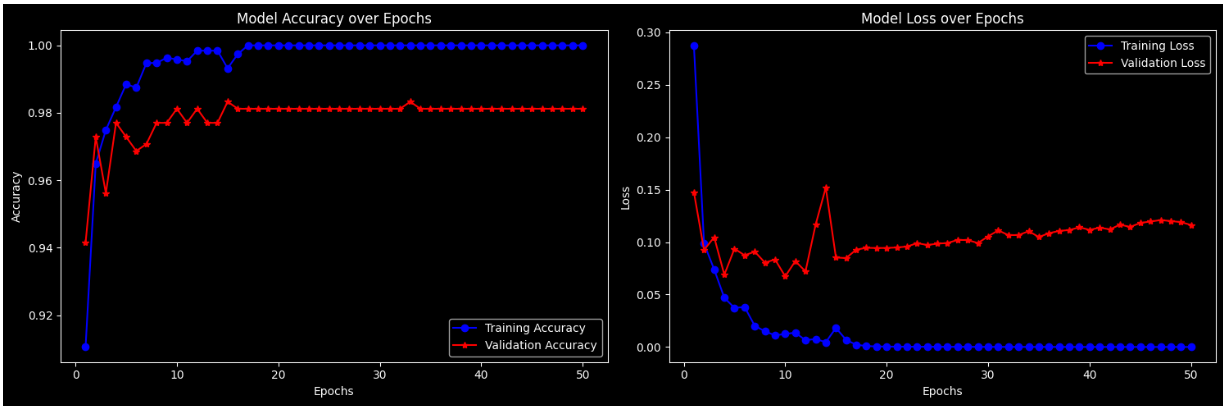
In [15]:
```
history_dict = history.history
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
epochs = range(1, len(accuracy) + 1)
```

In [16]:
```
plt.style.use('dark_background')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
ax1.plot(epochs, accuracy, 'bo-', label='Training Accuracy')
ax1.plot(epochs, val_accuracy, 'r*-', label='Validation Accuracy')
ax1.set_title('Model Accuracy over Epochs')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Accuracy')
ax1.legend()
```

```
ax2.plot(epochs, loss, 'bo-', label='Training Loss')
ax2.plot(epochs, val_loss, 'r*-', label='Validation Loss')
ax2.set_title('Model Loss over Epochs')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Loss')
ax2.legend()

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



In [17]:
```
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
label_mapping = {0: 'Low', 1: 'Medium', 2: 'High'}

predicted_labels = [label_mapping[label] for label in predicted_classes]
actual_labels = [label_mapping[label] for label in y_test]

correct_predictions = sum(1 for i in range(len(predicted_labels)) if predict
total_predictions = len(predicted_labels)

# print("Predictions vs Actual Values:")
# for i in range(total_predictions):
#     print(f"Predicted: {predicted_labels[i]}, Actual: {actual_labels[i]}")

print(f"\nTotal correct predictions: {correct_predictions} / {total_predicti
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step

Total correct predictions: 470 / 479
```

In [18]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion

accuracy = accuracy_score(y_test, predicted_classes)
print(f"\nModel Accuracy on Test Data: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, predicted_classes, target_names=['Low',
print("Confusion Matrix:")
print(confusion_matrix(y_test, predicted_classes))
```

```
Model Accuracy on Test Data: 0.98

Classification Report:
              precision    recall  f1-score   support

         Low       1.00      0.99      0.99       249
      Medium       0.97      0.99      0.98       214
        High       0.92      0.69      0.79        16

    accuracy                           0.98       479
   macro avg       0.96      0.89      0.92       479
weighted avg       0.98      0.98      0.98       479

Confusion Matrix:
[[247   2   0]
 [  1 212   1]
 [  0   5  11]]
```

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

---

In [19]:
```
X = data.drop(columns=['Profile'])
y = data['Profile']

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

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

model = Sequential()
model.add(Dense(128, input_shape=(X_train.shape[1],), activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', 

history = model.fit(X_train, y_train, epochs = 50, batch_size = 10, validati
```

```
/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
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.6180 - loss: 0.8518 -
val_accuracy: 0.9374 - val_loss: 0.2863
Epoch 2/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8758 - loss: 0.4162 -
val_accuracy: 0.9415 - val_loss: 0.2293
Epoch 3/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9085 - loss: 0.3039 -
val_accuracy: 0.9499 - val_loss: 0.1869
Epoch 4/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9215 - loss: 0.2616 -
val_accuracy: 0.9436 - val_loss: 0.1769
Epoch 5/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9318 - loss: 0.2804 -
val_accuracy: 0.9499 - val_loss: 0.1583
Epoch 6/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9484 - loss: 0.1957 -
val_accuracy: 0.9499 - val_loss: 0.1514
Epoch 7/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9551 - loss: 0.1607 -
val_accuracy: 0.9541 - val_loss: 0.1282
Epoch 8/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9470 - loss: 0.1840 -
val_accuracy: 0.9562 - val_loss: 0.1176
Epoch 9/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9527 - loss: 0.1487 -
val_accuracy: 0.9562 - val_loss: 0.1190
Epoch 10/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9617 - loss: 0.1338 -
val_accuracy: 0.9562 - val_loss: 0.1092
Epoch 11/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9452 - loss: 0.1666 -
val_accuracy: 0.9582 - val_loss: 0.1045
Epoch 12/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9528 - loss: 0.1366 -
val_accuracy: 0.9562 - val_loss: 0.0985
Epoch 13/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9552 - loss: 0.1429 -
val_accuracy: 0.9520 - val_loss: 0.1024
Epoch 14/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9650 - loss: 0.1059 -
val_accuracy: 0.9562 - val_loss: 0.0945
Epoch 15/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9663 - loss: 0.0962 -
val_accuracy: 0.9624 - val_loss: 0.0870
Epoch 16/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9669 - loss: 0.0960 -
val_accuracy: 0.9624 - val_loss: 0.0786
Epoch 17/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9652 - loss: 0.0846 -
val_accuracy: 0.9624 - val_loss: 0.0771
Epoch 18/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9683 - loss: 0.0832 -
val_accuracy: 0.9562 - val_loss: 0.0853
Epoch 19/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9675 - loss: 0.0893 -
```

```
val_accuracy: 0.9749 - val_loss: 0.0845
Epoch 20/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9643 - loss: 0.0998 -
val_accuracy: 0.9770 - val_loss: 0.0768
Epoch 21/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9683 - loss: 0.0827 -
val_accuracy: 0.9645 - val_loss: 0.0790
Epoch 22/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9709 - loss: 0.0792 -
val_accuracy: 0.9562 - val_loss: 0.1061
Epoch 23/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9697 - loss: 0.0955 -
val_accuracy: 0.9687 - val_loss: 0.0869
Epoch 24/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9666 - loss: 0.0892 -
val_accuracy: 0.9603 - val_loss: 0.0831
Epoch 25/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9695 - loss: 0.0870 -
val_accuracy: 0.9749 - val_loss: 0.0788
Epoch 26/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9678 - loss: 0.0668 -
val_accuracy: 0.9603 - val_loss: 0.0883
Epoch 27/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9689 - loss: 0.0656 -
val_accuracy: 0.9687 - val_loss: 0.0889
Epoch 28/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9684 - loss: 0.0620 -
val_accuracy: 0.9687 - val_loss: 0.0908
Epoch 29/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9766 - loss: 0.0558 -
val_accuracy: 0.9708 - val_loss: 0.0830
Epoch 30/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.9767 - loss: 0.0638
- val_accuracy: 0.9749 - val_loss: 0.0907
Epoch 31/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9786 - loss: 0.0434 -
val_accuracy: 0.9791 - val_loss: 0.0727
Epoch 32/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9762 - loss: 0.0560 -
val_accuracy: 0.9708 - val_loss: 0.0794
Epoch 33/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9744 - loss: 0.0665 -
val_accuracy: 0.9812 - val_loss: 0.0732
Epoch 34/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9829 - loss: 0.0589 -
val_accuracy: 0.9812 - val_loss: 0.0689
Epoch 35/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9785 - loss: 0.0494 -
val_accuracy: 0.9749 - val_loss: 0.0655
Epoch 36/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9792 - loss: 0.0545 -
val_accuracy: 0.9791 - val_loss: 0.0688
Epoch 37/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9735 - loss: 0.0643 -
val_accuracy: 0.9812 - val_loss: 0.0709
Epoch 38/50
```

```
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9826 - loss: 0.0426 -
val_accuracy: 0.9812 - val_loss: 0.0718
Epoch 39/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9766 - loss: 0.0600 -
val_accuracy: 0.9749 - val_loss: 0.0862
Epoch 40/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9886 - loss: 0.0329 -
val_accuracy: 0.9791 - val_loss: 0.0679
Epoch 41/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9853 - loss: 0.0439 -
val_accuracy: 0.9812 - val_loss: 0.0707
Epoch 42/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9858 - loss: 0.0323 -
val_accuracy: 0.9812 - val_loss: 0.0756
Epoch 43/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9903 - loss: 0.0299 -
val_accuracy: 0.9833 - val_loss: 0.0754
Epoch 44/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9865 - loss: 0.0339 -
val_accuracy: 0.9833 - val_loss: 0.0691
Epoch 45/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9851 - loss: 0.0428 -
val_accuracy: 0.9812 - val_loss: 0.0665
Epoch 46/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9875 - loss: 0.0358 -
val_accuracy: 0.9812 - val_loss: 0.0737
Epoch 47/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9809 - loss: 0.0449 -
val_accuracy: 0.9812 - val_loss: 0.0607
Epoch 48/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9878 - loss: 0.0354 -
val_accuracy: 0.9875 - val_loss: 0.0578
Epoch 49/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9829 - loss: 0.0428 -
val_accuracy: 0.9833 - val_loss: 0.0567
Epoch 50/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9854 - loss: 0.0387 -
val_accuracy: 0.9791 - val_loss: 0.0598
```

In [20]:

```
history_dict = history.history
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
epochs = range(1, len(accuracy) + 1)
```

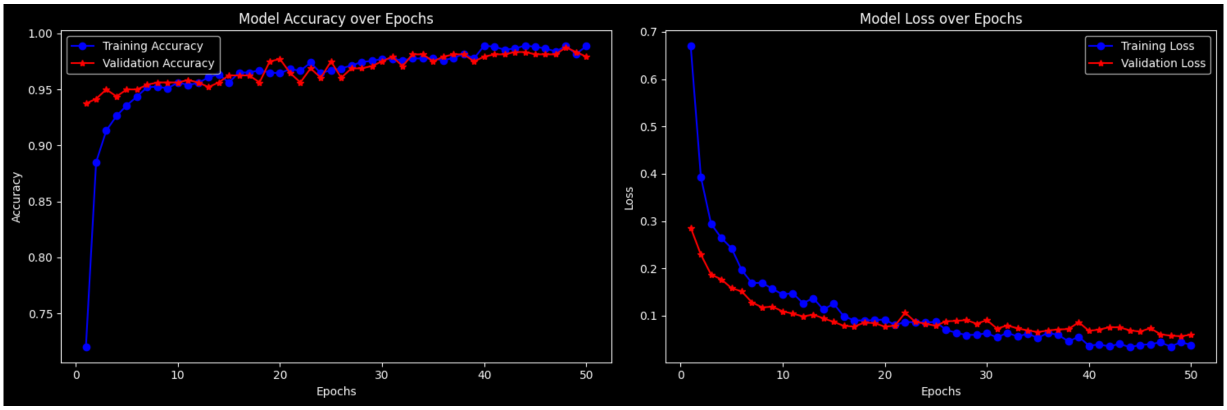
In [21]:

```
plt.style.use('dark_background')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
ax1.plot(epochs, accuracy, 'bo-', label='Training Accuracy')
ax1.plot(epochs, val_accuracy, 'r*-', label='Validation Accuracy')
ax1.set_title('Model Accuracy over Epochs')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Accuracy')
ax1.legend()
```

```
ax2.plot(epochs, loss, 'bo-', label='Training Loss')
ax2.plot(epochs, val_loss, 'r*-', label='Validation Loss')
ax2.set_title('Model Loss over Epochs')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Loss')
ax2.legend()

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



In [22]:
```
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
label_mapping = {0: 'Low', 1: 'Medium', 2: 'High'}

predicted_labels = [label_mapping[label] for label in predicted_classes]
actual_labels = [label_mapping[label] for label in y_test]

correct_predictions = sum(1 for i in range(len(predicted_labels)) if predict
total_predictions = len(predicted_labels)

# print("Predictions vs Actual Values:")
# for i in range(total_predictions):
#     print(f"Predicted: {predicted_labels[i]}, Actual: {actual_labels[i]}")

print(f"\nTotal correct predictions: {correct_predictions} / {total_predicti
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step

Total correct predictions: 469 / 479
```

In [23]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion

accuracy = accuracy_score(y_test, predicted_classes)
print(f"\nModel Accuracy on Test Data: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, predicted_classes, target_names=['Low',
print("Confusion Matrix:")
print(confusion_matrix(y_test, predicted_classes))
```

```
Model Accuracy on Test Data: 0.98

Classification Report:
              precision    recall  f1-score   support

         Low       1.00      0.99      0.99       249
      Medium       0.97      0.99      0.98       214
        High       0.85      0.69      0.76        16

    accuracy                           0.98       479
   macro avg       0.94      0.89      0.91       479
weighted avg       0.98      0.98      0.98       479

Confusion Matrix:
[[247   2   0]
 [  1 211   2]
 [  0   5  11]]
```

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

Tener cuidado aquí, ya que si las añadimos después nos da un rendimiento peor.

In [24]:
```
from tensorflow.keras.layers import BatchNormalization

X = data.drop(columns=['Profile'])
y = data['Profile']

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

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

model = Sequential()
model.add(Dense(128, input_shape=(X_train.shape[1],), activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(64, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

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

model.add(Dense(3, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',

history = model.fit(X_train, y_train, epochs = 50, batch_size = 10, validati
```

```
/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
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4367 - loss: 1.4128 -
val_accuracy: 0.8079 - val_loss: 0.5618
Epoch 2/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6875 - loss: 0.7240 -
val_accuracy: 0.9081 - val_loss: 0.3106
Epoch 3/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8342 - loss: 0.4589 -
val_accuracy: 0.9332 - val_loss: 0.2460
Epoch 4/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8288 - loss: 0.4446 -
val_accuracy: 0.9395 - val_loss: 0.2167
Epoch 5/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8229 - loss: 0.4637 -
val_accuracy: 0.9353 - val_loss: 0.2054
Epoch 6/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8371 - loss: 0.4257 -
val_accuracy: 0.9541 - val_loss: 0.1814
Epoch 7/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8650 - loss: 0.3924 -
val_accuracy: 0.9478 - val_loss: 0.1795
Epoch 8/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8826 - loss: 0.3254 -
val_accuracy: 0.9603 - val_loss: 0.1614
Epoch 9/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8686 - loss: 0.3353 -
val_accuracy: 0.9541 - val_loss: 0.1486
Epoch 10/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8695 - loss: 0.3211 -
val_accuracy: 0.9582 - val_loss: 0.1384
Epoch 11/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8842 - loss: 0.3131 -
val_accuracy: 0.9645 - val_loss: 0.1316
Epoch 12/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9101 - loss: 0.2555 -
val_accuracy: 0.9645 - val_loss: 0.1238
Epoch 13/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9044 - loss: 0.2690 -
val_accuracy: 0.9624 - val_loss: 0.1172
Epoch 14/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9103 - loss: 0.2660 -
val_accuracy: 0.9645 - val_loss: 0.1076
Epoch 15/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9104 - loss: 0.2502 -
val_accuracy: 0.9603 - val_loss: 0.0990
Epoch 16/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9000 - loss: 0.2833 -
val_accuracy: 0.9624 - val_loss: 0.0973
Epoch 17/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9236 - loss: 0.2237 -
val_accuracy: 0.9624 - val_loss: 0.0953
Epoch 18/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8981 - loss: 0.3195 -
val_accuracy: 0.9645 - val_loss: 0.1019
Epoch 19/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9265 - loss: 0.2348 -
```

```
val_accuracy: 0.9582 - val_loss: 0.0976
Epoch 20/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9264 - loss: 0.2252 -
val_accuracy: 0.9666 - val_loss: 0.0905
Epoch 21/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9261 - loss: 0.1954 -
val_accuracy: 0.9708 - val_loss: 0.0835
Epoch 22/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9292 - loss: 0.2051 -
val_accuracy: 0.9708 - val_loss: 0.0818
Epoch 23/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9245 - loss: 0.2149 -
val_accuracy: 0.9645 - val_loss: 0.0795
Epoch 24/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9233 - loss: 0.2020 -
val_accuracy: 0.9666 - val_loss: 0.0770
Epoch 25/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9297 - loss: 0.2099 -
val_accuracy: 0.9708 - val_loss: 0.0777
Epoch 26/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9285 - loss: 0.2052 -
val_accuracy: 0.9708 - val_loss: 0.0784
Epoch 27/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9226 - loss: 0.2180 -
val_accuracy: 0.9687 - val_loss: 0.0744
Epoch 28/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9347 - loss: 0.1914 -
val_accuracy: 0.9854 - val_loss: 0.0734
Epoch 29/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9205 - loss: 0.2307 -
val_accuracy: 0.9749 - val_loss: 0.0786
Epoch 30/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9328 - loss: 0.1927 -
val_accuracy: 0.9729 - val_loss: 0.0740
Epoch 31/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9297 - loss: 0.1977 -
val_accuracy: 0.9854 - val_loss: 0.0759
Epoch 32/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9291 - loss: 0.1856 -
val_accuracy: 0.9791 - val_loss: 0.0707
Epoch 33/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9271 - loss: 0.1964 -
val_accuracy: 0.9896 - val_loss: 0.0725
Epoch 34/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9321 - loss: 0.1918 -
val_accuracy: 0.9770 - val_loss: 0.0712
Epoch 35/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9357 - loss: 0.1679 -
val_accuracy: 0.9854 - val_loss: 0.0673
Epoch 36/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9344 - loss: 0.1826 -
val_accuracy: 0.9896 - val_loss: 0.0667
Epoch 37/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9281 - loss: 0.1899 -
val_accuracy: 0.9770 - val_loss: 0.0679
Epoch 38/50
```

```
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9290 - loss: 0.2068 -
val_accuracy: 0.9791 - val_loss: 0.0633
Epoch 39/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9254 - loss: 0.2109 -
val_accuracy: 0.9854 - val_loss: 0.0657
Epoch 40/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9352 - loss: 0.1976 -
val_accuracy: 0.9812 - val_loss: 0.0622
Epoch 41/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9389 - loss: 0.1989 -
val_accuracy: 0.9791 - val_loss: 0.0645
Epoch 42/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9433 - loss: 0.1567 -
val_accuracy: 0.9812 - val_loss: 0.0652
Epoch 43/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9423 - loss: 0.1612 -
val_accuracy: 0.9812 - val_loss: 0.0663
Epoch 44/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9464 - loss: 0.1452 -
val_accuracy: 0.9791 - val_loss: 0.0657
Epoch 45/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9489 - loss: 0.1489 -
val_accuracy: 0.9854 - val_loss: 0.0615
Epoch 46/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9298 - loss: 0.2247 -
val_accuracy: 0.9812 - val_loss: 0.0620
Epoch 47/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9427 - loss: 0.1473 -
val_accuracy: 0.9812 - val_loss: 0.0638
Epoch 48/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9338 - loss: 0.1767 -
val_accuracy: 0.9812 - val_loss: 0.0600
Epoch 49/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9435 - loss: 0.1819 -
val_accuracy: 0.9833 - val_loss: 0.0575
Epoch 50/50
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9537 - loss: 0.1444 -
val_accuracy: 0.9854 - val_loss: 0.0600
```

In [25]:

```python
history_dict = history.history
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']
epochs = range(1, len(accuracy) + 1)
```

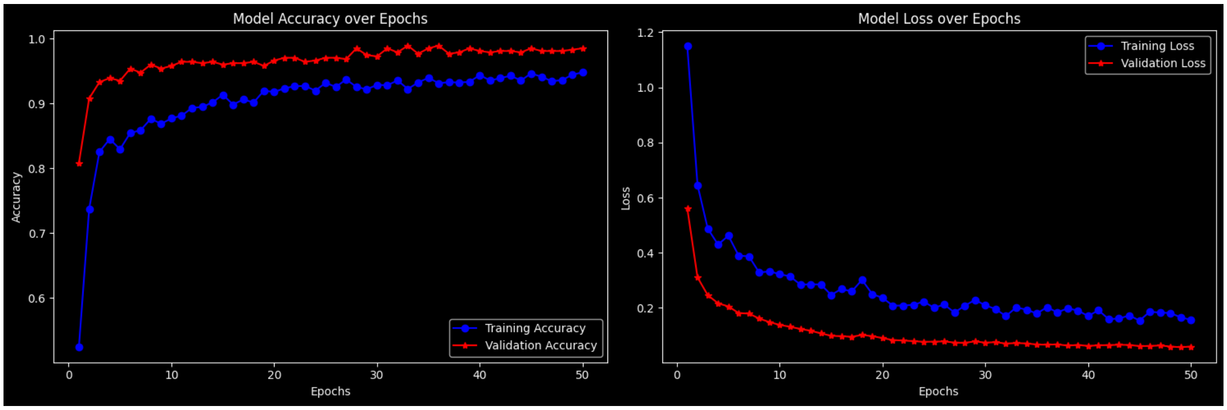
In [26]:

```python
plt.style.use('dark_background')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
ax1.plot(epochs, accuracy, 'bo-', label='Training Accuracy')
ax1.plot(epochs, val_accuracy, 'r*-', label='Validation Accuracy')
ax1.set_title('Model Accuracy over Epochs')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Accuracy')
ax1.legend()
```

```
ax2.plot(epochs, loss, 'bo-', label='Training Loss')
ax2.plot(epochs, val_loss, 'r*-', label='Validation Loss')
ax2.set_title('Model Loss over Epochs')
ax2.set_xlabel('Epochs')
ax2.set_ylabel('Loss')
ax2.legend()

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



In [27]:
```
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
label_mapping = {0: 'Low', 1: 'Medium', 2: 'High'}

predicted_labels = [label_mapping[label] for label in predicted_classes]
actual_labels = [label_mapping[label] for label in y_test]

correct_predictions = sum(1 for i in range(len(predicted_labels)) if predict
total_predictions = len(predicted_labels)

# print("Predictions vs Actual Values:")
# for i in range(total_predictions):
#     print(f"Predicted: {predicted_labels[i]}, Actual: {actual_labels[i]}")

print(f"\nTotal correct predictions: {correct_predictions} / {total_predicti
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step

Total correct predictions: 472 / 479
```

In [28]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion

accuracy = accuracy_score(y_test, predicted_classes)
print(f"\nModel Accuracy on Test Data: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, predicted_classes, target_names=['Low',
print("Confusion Matrix:")
print(confusion_matrix(y_test, predicted_classes))
```

```
Model Accuracy on Test Data: 0.99

Classification Report:
              precision    recall  f1-score   support

         Low       0.99      0.99      0.99       249
      Medium       0.98      0.99      0.98       214
        High       0.93      0.88      0.90        16

    accuracy                           0.99       479
   macro avg       0.97      0.95      0.96       479
weighted avg       0.99      0.99      0.99       479

Confusion Matrix:
[[247   2   0]
 [  2 211   1]
 [  0   2  14]]
```

Comparative Table

Model	Total Correct Predictions
Experiment 1: A single Dense Hidden Layer	471 / 479
Experiment 2: A set of three Dense Hidden Layers	470 / 479
Experiment 3: A dropout layer after each Dense Hidden Layer	473 / 479
Experiment 4: A Batch Normalization Layer before each Dropout Layer	469 / 479