In [1]:

```
#Importing dataset

import pandas as pd
hr_df = pd.read_csv("D:\Data Science and Deep Learning for Business\datascienceforbu

print(hr_df.info())
print("\n",hr_df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   employee_id            14999 non-null  int64
 1   number_project         14999 non-null  int64
 2   average_montly_hours   14999 non-null  int64
 3   time_spend_company     14999 non-null  int64
 4   Work_accident          14999 non-null  int64
 5   left                   14999 non-null  int64
 6   promotion_last_5years  14999 non-null  int64
 7   department             14999 non-null  object
 8   salary                 14999 non-null  object
dtypes: int64(7), object(2)
memory usage: 1.0+ MB
None

 (14999, 9)
```

In [2]:

```
#view categorical data and viewing their unique value

print(hr_df.select_dtypes(exclude=['int', 'float']).columns)
print("\n",hr_df['department'].unique())
print("\n",hr_df['salary'].unique())
```

```
Index(['department', 'salary'], dtype='object')

 ['sales' 'accounting' 'hr' 'technical' 'support' 'management' 'IT'
 'product_mng' 'marketing' 'RandD']

 ['low' 'medium' 'high']
```

In [3]:

```
#loading employee satisfication dataset

emp_stats = pd.read_excel("D:\Data Science and Deep Learning for Business\datascienc
emp_stats.head()
```

Out[3]:

	EMPLOYEE #	satisfaction_level	last_evaluation
0	1003	0.38	0.53
1	1005	0.80	0.86
2	1486	0.11	0.88
3	1038	0.72	0.87
4	1057	0.37	0.52

In [4]:

```
#joining two datasets based on employee id column
main_df = hr_df.set_index('employee_id').join(emp_stats.set_index('EMPLOYEE #'))
main_df.head()
```

Out[4]:

	number_project	average_montly_hours	time_spend_company	Work_accident	l
employee_id					
1003	2	157	3	0	
1005	5	262	6	0	
1486	7	272	4	0	
1038	5	223	5	0	
1057	2	159	3	0	

In [5]:

```
#flattening the dataset after merge
main_df = main_df.reset_index()
main_df.head()
```

Out[5]:

	employee_id	number_project	average_montly_hours	time_spend_company	Work_accident
0	1003	2	157	3	0
1	1005	5	262	6	0
2	1486	7	272	4	0
3	1038	5	223	5	0
4	1057	2	159	3	0

In [6]:

```
#checking any NULL values or missing values
print(main_df.isnull().sum())
#viewing null value records
main_df[main_df.isnull().any(axis=1)]


```

```
employee_id               0
number_project            0
average_montly_hours      0
time_spend_company        0
Work_accident             0
left                      0
promotion_last_5years     0
department                0
salary                    0
satisfaction_level       27
last_evaluation          27
dtype: int64
```

Out[6]:

	employee_id	number_project	average_montly_hours	time_spend_company	Work_acc
18	3794	2	160	3	
19	1140	5	262	5	
33	1230	2	140	3	
53	1340	2	132	3	
72	22316	2	149	3	
92	1581	2	143	3	
107	17376	2	148	3	
120	1739	4	158	4	
137	1847	2	129	3	
175	32923	4	164	2	
191	2160	4	226	6	
352	3150	4	262	6	
376	3250	4	296	2	
402	3405	5	275	5	
427	78130	3	180	4	
442	3635	5	229	5	
468	3755	5	245	5	
543	4150	5	237	5	
892	43615	4	276	5	
1588	42185	5	264	5	
1934	11895	4	225	5	
2343	14170	3	115	2	
2743	16445	5	149	2	
3170	18980	5	186	2	
3609	21580	3	263	2	
3776	22555	4	214	3	

In [7]:

```
#filling missing value with mean values
main_df.fillna(main_df.mean(numeric_only=True),inplace=True)
main_df.head()
```

Out[7]:

	employee_id	number_project	average_montly_hours	time_spend_company	Work_accident	
0	1003	2	157	3	0	
1	1005	5	262	6	0	
2	1486	7	272	4	0	
3	1038	5	223	5	0	
4	1057	2	159	3	0	

4122	24505	3	192	3	3	
4740	27950	3	253	3	3	
5028	29640	4	180	4	4	
6453	38090	5	166	2	2	
7005	41535	4	150	3	3	
8630	50960	4	167	3		
9455	55770	4	270	3		
9901	58630	5	252	3		
10647	62595	4	165	3		
10962	64350	5	233	3		

	employee_id	number_project	average_montly_hours	time_spend_company	Work_acc
11575	20215	3	192	7	
1967	70005	3	148	3	
12422	72475	5	257	5	
12853	74880	6	136	2	
13482	78780	3	207	7	
0	11428	81315	3	133	3

```
1     3571
Name: left, dtype: int64
```

In [9]:

```
# Perform One Hot Encoding on Categorical Data

categorial = ['department','salary']
main_df_final = pd.get_dummies(main_df_final, columns=categorial, drop_first=True)
main_df_final.head()

```

Out[9]:

	number_project	average_montly_hours	time_spend_company	Work_accident	left	promot
0	2	157	3	0	1	
1	5	262	6	0	1	
2	7	272	4	0	1	
3	5	223	5	0	1	
4	2	159	3	0	1	

In [10]:

```
#splitting the data

from sklearn.model_selection import train_test_split

#Removing target column
X = main_df_final.drop(['left'],axis=1).values
# Assigning target column
y = main_df_final['left'].values
# Splittng the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [11]:

```
# Normalize the data
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [12]:

```
#Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

model = LogisticRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print("Accuracy {0:.2f}%".format(100*accuracy_score(predictions, y_test)))
print("\nConfusion Matrix\n",confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))


```

```
Accuracy 78.47%

Confusion Matrix
 [[3142  249]
 [ 720  389]]
              precision    recall  f1-score   support

           0       0.81      0.93      0.87      3391
           1       0.61      0.35      0.45      1109

    accuracy                           0.78      4500
   macro avg       0.71      0.64      0.66      4500
weighted avg       0.76      0.78      0.76      4500
```

In [13]:

```
#Deep Learning
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import Dropout

model2 = Sequential()

# Hidden Layer 1
model2.add(Dense(270, activation='relu', input_dim=18, kernel_regularizer=l2(0.01)))
model2.add(Dropout(0.3))

# Hidden Layer 1
model2.add(Dense(180, activation='relu', input_dim=18, kernel_regularizer=l2(0.01)))
model2.add(Dropout(0.3))

# Hidden Layer 2
model2.add(Dense(90, activation = 'relu',input_dim=18,  kernel_regularizer=l2(0.01)))
model2.add(Dropout(0.3))


model2.add(Dense(1, activation='sigmoid'))

model2.summary()

model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 270)               5130

 dropout (Dropout)           (None, 270)               0

 dense_1 (Dense)             (None, 180)               48780

 dropout_1 (Dropout)         (None, 180)               0

 dense_2 (Dense)             (None, 90)                16290

 dropout_2 (Dropout)         (None, 90)                0

 dense_3 (Dense)             (None, 1)                 91

=================================================================
Total params: 70,291
Trainable params: 70,291
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
# training
batch_size = 10
epochs = 25

history = model2.fit(X_train,
                     y_train,
                     batch_size = batch_size,
                     epochs = epochs,
                     verbose = 1,
                     validation_data = (X_test, y_test))

score = model2.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Epoch 1/25
1050/1050 [==============================] - 4s 3ms/step - loss: 0.6867 -
accuracy: 0.9007 - val_loss: 0.2846 - val_accuracy: 0.9498
Epoch 2/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2869 -
accuracy: 0.9377 - val_loss: 0.2444 - val_accuracy: 0.9507
Epoch 3/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2734 -
accuracy: 0.9380 - val_loss: 0.2374 - val_accuracy: 0.9524
Epoch 4/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2638 -
accuracy: 0.9403 - val_loss: 0.2247 - val_accuracy: 0.9593
Epoch 5/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2578 -
accuracy: 0.9412 - val_loss: 0.2338 - val_accuracy: 0.9476
Epoch 6/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2582 -
accuracy: 0.9409 - val_loss: 0.2297 - val_accuracy: 0.9507
Epoch 7/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2546 -
accuracy: 0.9412 - val_loss: 0.2206 - val_accuracy: 0.9576
Epoch 8/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2491 -
accuracy: 0.9412 - val_loss: 0.2176 - val_accuracy: 0.9544
Epoch 9/25
1050/1050 [==============================] - 5s 4ms/step - loss: 0.2492 -
accuracy: 0.9445 - val_loss: 0.2182 - val_accuracy: 0.9587
Epoch 10/25
1050/1050 [==============================] - 5s 4ms/step - loss: 0.2430 -
accuracy: 0.9433 - val_loss: 0.2157 - val_accuracy: 0.9529
Epoch 11/25
1050/1050 [==============================] - 5s 4ms/step - loss: 0.2433 -
accuracy: 0.9445 - val_loss: 0.2073 - val_accuracy: 0.9580
Epoch 12/25
1050/1050 [==============================] - 5s 4ms/step - loss: 0.2406 -
accuracy: 0.9446 - val_loss: 0.2059 - val_accuracy: 0.9567
Epoch 13/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2407 -
accuracy: 0.9429 - val_loss: 0.2098 - val_accuracy: 0.9531
Epoch 14/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2388 -
accuracy: 0.9436 - val_loss: 0.2177 - val_accuracy: 0.9518
Epoch 15/25
1050/1050 [==============================] - 5s 4ms/step - loss: 0.2386 -
accuracy: 0.9454 - val_loss: 0.2006 - val_accuracy: 0.9589
Epoch 16/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2371 -
accuracy: 0.9436 - val_loss: 0.2088 - val_accuracy: 0.9589
Epoch 17/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2397 -
accuracy: 0.9444 - val_loss: 0.2016 - val_accuracy: 0.9587
Epoch 18/25
1050/1050 [==============================] - 5s 5ms/step - loss: 0.2384 -
accuracy: 0.9436 - val_loss: 0.2034 - val_accuracy: 0.9584
Epoch 19/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2342 -
accuracy: 0.9459 - val_loss: 0.1956 - val_accuracy: 0.9596
Epoch 20/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2315 -
accuracy: 0.9480 - val_loss: 0.2192 - val_accuracy: 0.9529
Epoch 21/25
```

```
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2353 -
accuracy: 0.9445 - val_loss: 0.1934 - val_accuracy: 0.9631
Epoch 22/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2310 -
accuracy: 0.9469 - val_loss: 0.1924 - val_accuracy: 0.9622
Epoch 23/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2338 -
accuracy: 0.9453 - val_loss: 0.1953 - val_accuracy: 0.9618
Epoch 24/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2318 -
accuracy: 0.9457 - val_loss: 0.1960 - val_accuracy: 0.9598
Epoch 25/25
1050/1050 [==============================] - 4s 4ms/step - loss: 0.2286 -
accuracy: 0.9470 - val_loss: 0.2119 - val_accuracy: 0.9511
Test loss: 0.2119394838809967
Test accuracy: 0.9511111378669739
```

In [15]:

```
predictions = model2.predict(X_test)
predictions = (predictions > 0.5)

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
141/141 [==============================] - 0s 2ms/step
[[3334   57]
 [ 163  946]]
              precision    recall  f1-score   support

           0       0.95      0.98      0.97      3391
           1       0.94      0.85      0.90      1109

    accuracy                           0.95      4500
   macro avg       0.95      0.92      0.93      4500
weighted avg       0.95      0.95      0.95      4500
```

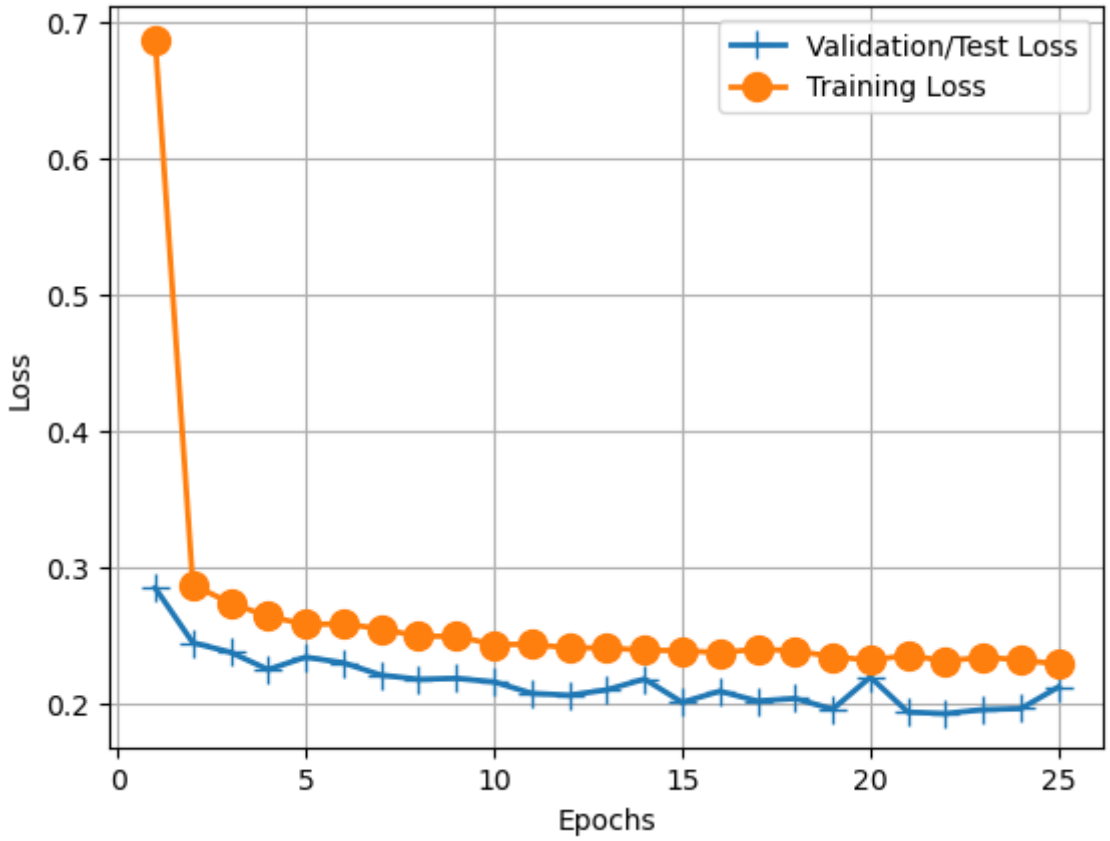
In [16]:

```
# loss charts
import matplotlib.pyplot as plt

history_dict = history.history

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)

line1 = plt.plot(epochs, val_loss_values, label='Validation/Test Loss')
line2 = plt.plot(epochs, loss_values, label='Training Loss')
plt.setp(line1, linewidth=2.0, marker = '+', markersize=10.0)
plt.setp(line2, linewidth=2.0, marker = 'o', markersize=10.0)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()
plt.show()
```



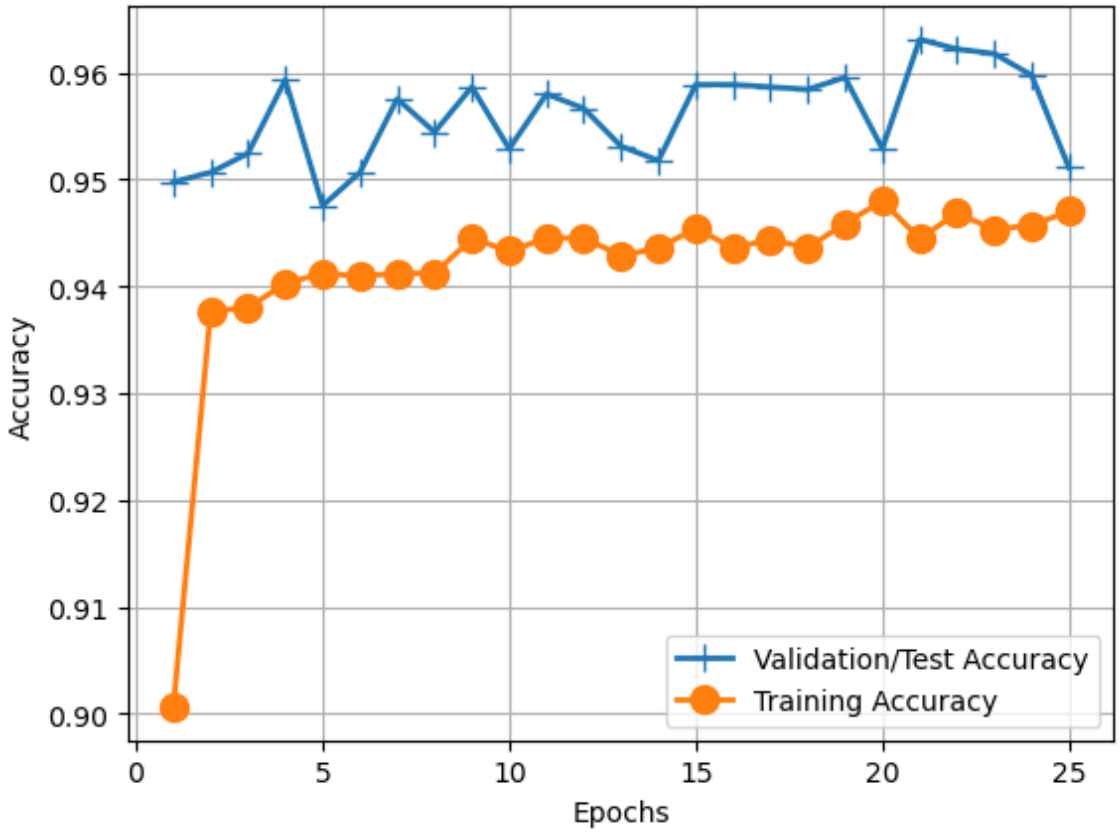
In [17]:

```
# accuracy charts
import matplotlib.pyplot as plt

history_dict = history.history

acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
epochs = range(1, len(loss_values) + 1)

line1 = plt.plot(epochs, val_acc_values, label='Validation/Test Accuracy')
line2 = plt.plot(epochs, acc_values, label='Training Accuracy')
plt.setp(line1, linewidth=2.0, marker = '+', markersize=10.0)
plt.setp(line2, linewidth=2.0, marker = 'o', markersize=10.0)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()
plt.show()
```



In [ ]:

```
1
```