2. Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset

In [50]:
```
import tensorflow as tf
from tensorflow.keras import Sequential  # type: ignore
from tensorflow.keras.layers import Dense, Input, Dropout, Flatten # type: ignore
from tensorflow.keras.utils import to_categorical # type: ignore
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import pandas as pd
```

In [51]:
```
import requests

url = 'https://archive.ics.uci.edu/static/public/59/letter+recognition.zip'
filename = 'letter+recognition.zip'

response = requests.get(url)
with open(filename, 'wb') as f:
    f.write(response.content)
```

In [ ]:
```
import zipfile
# this is optional instead just extract the zip folder downloaded in the dir

with zipfile.ZipFile(filename, 'r') as zip_ref:
    zip_ref.extractall('letter_recognition')
```

In [53]:
```
df = pd.read_csv('./letter_recognition/letter-recognition.data', header=None)
```

In [54]:
```
df.head()
```

Out[54]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

In [55]:
```
df[0] = df[0].apply(lambda x: ord(x) - ord('A'))
```

In [56]:
```
df.head()
```

Out[56]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	19	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	8	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	3	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	13	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	6	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

In [57]:
```
X = df.iloc[:, 1:].values
y = df.iloc[:, 0].values
```

In [58]:
```
y = to_categorical(y, num_classes=26) #one-hot encoding
```

In [59]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [60]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [61]:
```
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(26, activation='softmax')
])
```

In [62]:
```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [63]:
```
history = model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)
```

```
Epoch 1/30
400/400 [==============================] - 1s 1ms/step - loss: 2.0862 - accuracy: 0.3950 - val
_loss: 1.1743 - val_accuracy: 0.6875
Epoch 2/30
400/400 [==============================] - 0s 1ms/step - loss: 1.2530 - accuracy: 0.6119 - val
_loss: 0.8957 - val_accuracy: 0.7544
Epoch 3/30
400/400 [==============================] - 0s 1ms/step - loss: 1.0534 - accuracy: 0.6752 - val
_loss: 0.7655 - val_accuracy: 0.7847
Epoch 4/30
400/400 [==============================] - 0s 1ms/step - loss: 0.9591 - accuracy: 0.7011 - val
_loss: 0.6893 - val_accuracy: 0.8028
Epoch 5/30
400/400 [==============================] - 0s 1ms/step - loss: 0.8698 - accuracy: 0.7261 - val
_loss: 0.6219 - val_accuracy: 0.8272
Epoch 6/30
400/400 [==============================] - 0s 1ms/step - loss: 0.8046 - accuracy: 0.7511 - val
_loss: 0.5804 - val_accuracy: 0.8363
Epoch 7/30
400/400 [==============================] - 0s 1ms/step - loss: 0.7686 - accuracy: 0.7559 - val
_loss: 0.5389 - val_accuracy: 0.8450
Epoch 8/30
400/400 [==============================] - 0s 1ms/step - loss: 0.7332 - accuracy: 0.7687 - val
_loss: 0.5101 - val_accuracy: 0.8484
Epoch 9/30
400/400 [==============================] - 0s 1ms/step - loss: 0.6948 - accuracy: 0.7787 - val
_loss: 0.4822 - val_accuracy: 0.8597
Epoch 10/30
400/400 [==============================] - 0s 1ms/step - loss: 0.6634 - accuracy: 0.7880 - val
_loss: 0.4511 - val_accuracy: 0.8709
Epoch 11/30
400/400 [==============================] - 0s 1ms/step - loss: 0.6445 - accuracy: 0.7914 - val
_loss: 0.4297 - val_accuracy: 0.8744
Epoch 12/30
400/400 [==============================] - 0s 1ms/step - loss: 0.6186 - accuracy: 0.7991 - val
_loss: 0.4114 - val_accuracy: 0.8800
Epoch 13/30
400/400 [==============================] - 0s 1ms/step - loss: 0.6187 - accuracy: 0.8035 - val
_loss: 0.4022 - val_accuracy: 0.8825
Epoch 14/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5823 - accuracy: 0.8111 - val
_loss: 0.3812 - val_accuracy: 0.8891
Epoch 15/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5753 - accuracy: 0.8159 - val
_loss: 0.3706 - val_accuracy: 0.8906
Epoch 16/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5486 - accuracy: 0.8273 - val
_loss: 0.3590 - val_accuracy: 0.8941
Epoch 17/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5380 - accuracy: 0.8272 - val
_loss: 0.3443 - val_accuracy: 0.8969
Epoch 18/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5389 - accuracy: 0.8298 - val
_loss: 0.3343 - val_accuracy: 0.9047
Epoch 19/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5236 - accuracy: 0.8298 - val
_loss: 0.3346 - val_accuracy: 0.8997
Epoch 20/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5088 - accuracy: 0.8360 - val
_loss: 0.3182 - val_accuracy: 0.9028
Epoch 21/30
400/400 [==============================] - 0s 1ms/step - loss: 0.5092 - accuracy: 0.8352 - val
_loss: 0.3208 - val_accuracy: 0.9075
```

```
Epoch 22/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4935 - accuracy: 0.8413 - val
_loss: 0.3069 - val_accuracy: 0.9059
Epoch 23/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4877 - accuracy: 0.8420 - val
_loss: 0.3053 - val_accuracy: 0.9112
Epoch 24/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4898 - accuracy: 0.8405 - val
_loss: 0.2994 - val_accuracy: 0.9091
Epoch 25/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4641 - accuracy: 0.8509 - val
_loss: 0.2921 - val_accuracy: 0.9144
Epoch 26/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4555 - accuracy: 0.8528 - val
_loss: 0.2858 - val_accuracy: 0.9078
Epoch 27/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4558 - accuracy: 0.8510 - val
_loss: 0.2792 - val_accuracy: 0.9144
Epoch 28/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4598 - accuracy: 0.8494 - val
_loss: 0.2825 - val_accuracy: 0.9106
Epoch 29/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4494 - accuracy: 0.8528 - val
_loss: 0.2746 - val_accuracy: 0.9153
Epoch 30/30
400/400 [==============================] - 0s 1ms/step - loss: 0.4539 - accuracy: 0.8512 - val
_loss: 0.2749 - val_accuracy: 0.9144
```

In [65]:
```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
```

```
Test Loss: 0.2455
Test Accuracy: 92.55%
```

In [66]:
```
predictions = model.predict(X_test)
y_pred = predictions.argmax(axis=1)
y_true = y_test.argmax(axis=1)
```

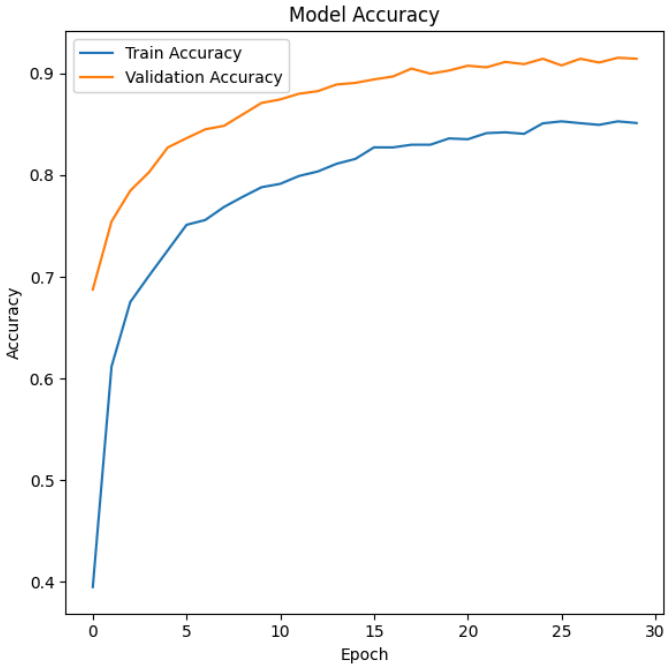
```
125/125 [==============================] - 0s 567us/step
```

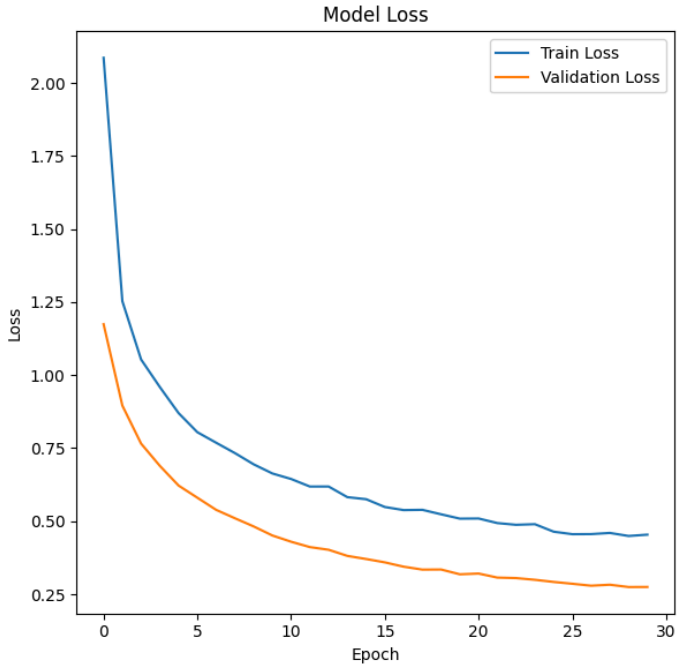
In [70]:
```
print("classification Report:\n")
print(classification_report(y_true, y_pred, target_names=[chr(i) for i in range(ord('A'), ord
```

```
classification Report:

              precision    recall  f1-score   support

           A       0.95      0.97      0.96       149
           B       0.81      0.94      0.87       153
           C       0.99      0.91      0.95       137
           D       0.88      0.94      0.91       156
           E       0.88      0.93      0.90       141
           F       0.89      0.90      0.90       140
           G       0.89      0.94      0.91       160
           H       0.92      0.78      0.84       144
           I       0.91      0.92      0.92       146
           J       0.96      0.91      0.93       149
           K       0.93      0.88      0.91       130
           L       0.98      0.90      0.94       155
           M       0.98      0.96      0.97       168
           N       0.99      0.91      0.94       151
           O       0.89      0.92      0.90       145
           P       0.95      0.91      0.93       173
           Q       0.98      0.98      0.98       166
           R       0.80      0.91      0.85       160
           S       0.93      0.91      0.92       171
           T       0.96      0.93      0.95       163
           U       0.93      0.95      0.94       183
           V       0.95      0.94      0.95       158
           W       0.93      0.97      0.95       148
           X       0.94      0.98      0.96       154
           Y       0.98      0.93      0.95       168
           Z       0.94      0.92      0.93       132

    accuracy                           0.93      4000
   macro avg       0.93      0.92      0.93      4000
weighted avg       0.93      0.93      0.93      4000
```

In [71]:
```
model.save('ocr_multiclass_model.keras')
```

In [73]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()

plt.show()
```


Model Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.4
0.5
0.6
0.7
0.8
0.9
0
5
10
15
20
25
30


Model Loss
Train Loss
Validation Loss
Loss
Epoch
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
5
10
15
20
25
30

In [ ]: