Multiclass Classification using Deep Neural Networks

(OCR Letter Recognition Dataset)

In [1]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import pandas as pd
```

In [2]:
```
import requests
import zipfile
import os

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)

with zipfile.ZipFile(filename, 'r') as zip_ref:
    zip_ref.extractall('letter_recognition')  # Specify the directory to extract files to
```

In [3]:
```
extracted_folder = 'letter_recognition'
extracted_files = os.listdir(extracted_folder)
print(extracted_files)
```

```
['Index', 'letter-recognition.data', 'letter-recognition.data.Z', 'letter-recognition.names']
```

In [4]:
```
csv_file = os.path.join(extracted_folder, 'letter-recognition.data')
```

In [5]:
```
df = pd.read_csv(csv_file, header=None)
```

In [6]:
```
df.head()
```

Out[6]:

	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 [7]:
```
df[0] = df[0].apply(lambda x: ord(x) - ord('A'))
#Convert the letter labels to numerical values.
```

In [8]:
```
X = df.iloc[:, 1:].values
y = df.iloc[:, 0].values
```

In [9]:
```
y = to_categorical(y, num_classes=26)
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [11]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [12]:
```
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 [13]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [14]:
```
history = model.fit(X_train, y_train,
                    epochs=30,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=1)
```

```
Epoch 1/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2519 - loss: 2.6004 - val_accuracy: 0.6812 - val_loss: 1.1776
Epoch 2/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5847 - loss: 1.3253 - val_accuracy: 0.7522 - val_loss: 0.8968
Epoch 3/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6500 - loss: 1.1168 - val_accuracy: 0.7800 - val_loss: 0.7649
Epoch 4/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6938 - loss: 0.9873 - val_accuracy: 0.8022 - val_loss: 0.6911
Epoch 5/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7176 - loss: 0.8903 - val_accuracy: 0.8213 - val_loss: 0.6220
Epoch 6/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7454 - loss: 0.8174 - val_accuracy: 0.8347 - val_loss: 0.5741
Epoch 7/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7587 - loss: 0.7614 - val_accuracy: 0.8544 - val_loss: 0.5321
Epoch 8/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7665 - loss: 0.7400 - val_accuracy: 0.8609 - val_loss: 0.5010
Epoch 9/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7755 - loss: 0.7094 - val_accuracy: 0.8650 - val_loss: 0.4722
Epoch 10/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7853 - loss: 0.6757 - val_accuracy: 0.8797 - val_loss: 0.4507
Epoch 11/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7955 - loss: 0.6306 - val_accuracy: 0.8788 - val_loss: 0.4290
Epoch 12/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7889 - loss: 0.6434 - val_accuracy: 0.8834 - val_loss: 0.4108
Epoch 13/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8053 - loss: 0.6070 - val_accuracy: 0.8897 - val_loss: 0.3890
Epoch 14/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8032 - loss: 0.6047 - val_accuracy: 0.8944 - val_loss: 0.3747
Epoch 15/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8113 - loss: 0.5747 - val_accuracy: 0.8950 - val_loss: 0.3637
Epoch 16/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8195 - loss: 0.5663 - val_accuracy: 0.8988 - val_loss: 0.3519
Epoch 17/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8166 - loss: 0.5550 - val_accuracy: 0.9019 - val_loss: 0.3399
Epoch 18/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8245 - loss: 0.5376 - val_accuracy: 0.9031 - val_loss: 0.3321
Epoch 19/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8289 - loss: 0.5208 - val_accuracy: 0.9097 - val_loss: 0.3190
Epoch 20/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8326 - loss: 0.5101 - val_accuracy: 0.9075 - val_loss: 0.3153
Epoch 21/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8417 - loss: 0.4938 - val_accuracy: 0.9119 - val_loss: 0.3049
Epoch 22/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8387 - loss: 0.4967 - val_accuracy: 0.9153 - val_loss: 0.2923
Epoch 23/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8380 - loss: 0.4852 - val_accuracy: 0.9169 - val_loss: 0.2926
Epoch 24/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8417 - loss: 0.4810 - val_accuracy: 0.9175 - val_loss: 0.2874
Epoch 25/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8432 - loss: 0.4705 - val_accuracy: 0.9162 - val_loss: 0.2860
Epoch 26/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8472 - loss: 0.4641 - val_accuracy: 0.9234 - val_loss: 0.2786
Epoch 27/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8514 - loss: 0.4454 - val_accuracy: 0.9194 - val_loss: 0.2757
Epoch 28/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8505 - loss: 0.4744 - val_accuracy: 0.9222 - val_loss: 0.2705
Epoch 29/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8506 - loss: 0.4503 - val_accuracy: 0.9247 - val_loss: 0.2624
Epoch 30/30
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8480 - loss: 0.4572 - val_accuracy: 0.9244 - val_loss: 0.2584
```

In [15]:
```
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: 0.2556
Test Accuracy: 92.05%
```

In [16]:
```
predictions = model.predict(X_test)
y_pred = predictions.argmax(axis=1)
y_true = y_test.argmax(axis=1)
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 857us/step
```

In [17]:
```
print("Classification Report:\n")
print(classification_report(y_true, y_pred, target_names=[chr(i) for i in range(ord('A'), ord('Z')+1)]))
```

```
Classification Report:

              precision    recall  f1-score   support

           A       0.94      0.99      0.96       149
           B       0.86      0.93      0.89       153
           C       0.94      0.90      0.92       137
           D       0.87      0.93      0.90       156
           E       0.92      0.92      0.92       141
           F       0.84      0.94      0.89       140
           G       0.86      0.91      0.88       160
           H       0.94      0.75      0.83       144
           I       0.94      0.92      0.93       146
           J       0.96      0.92      0.94       149
           K       0.86      0.83      0.84       130
           L       0.99      0.93      0.96       155
           M       0.96      0.96      0.96       168
           N       0.97      0.91      0.94       151
           O       0.87      0.90      0.89       145
           P       0.98      0.85      0.91       173
           Q       0.96      0.95      0.95       166
           R       0.76      0.93      0.83       160
           S       0.96      0.93      0.95       171
           T       0.96      0.91      0.93       163
           U       0.95      0.94      0.94       183
           V       0.99      0.91      0.95       158
           W       0.91      0.97      0.94       148
           X       0.92      0.99      0.95       154
           Y       0.98      0.98      0.98       168
           Z       0.92      0.92      0.92       132

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

(optional code below)

In [18]:

```
model.save("ocr_multiclass_model.keras")
```

In [19]:

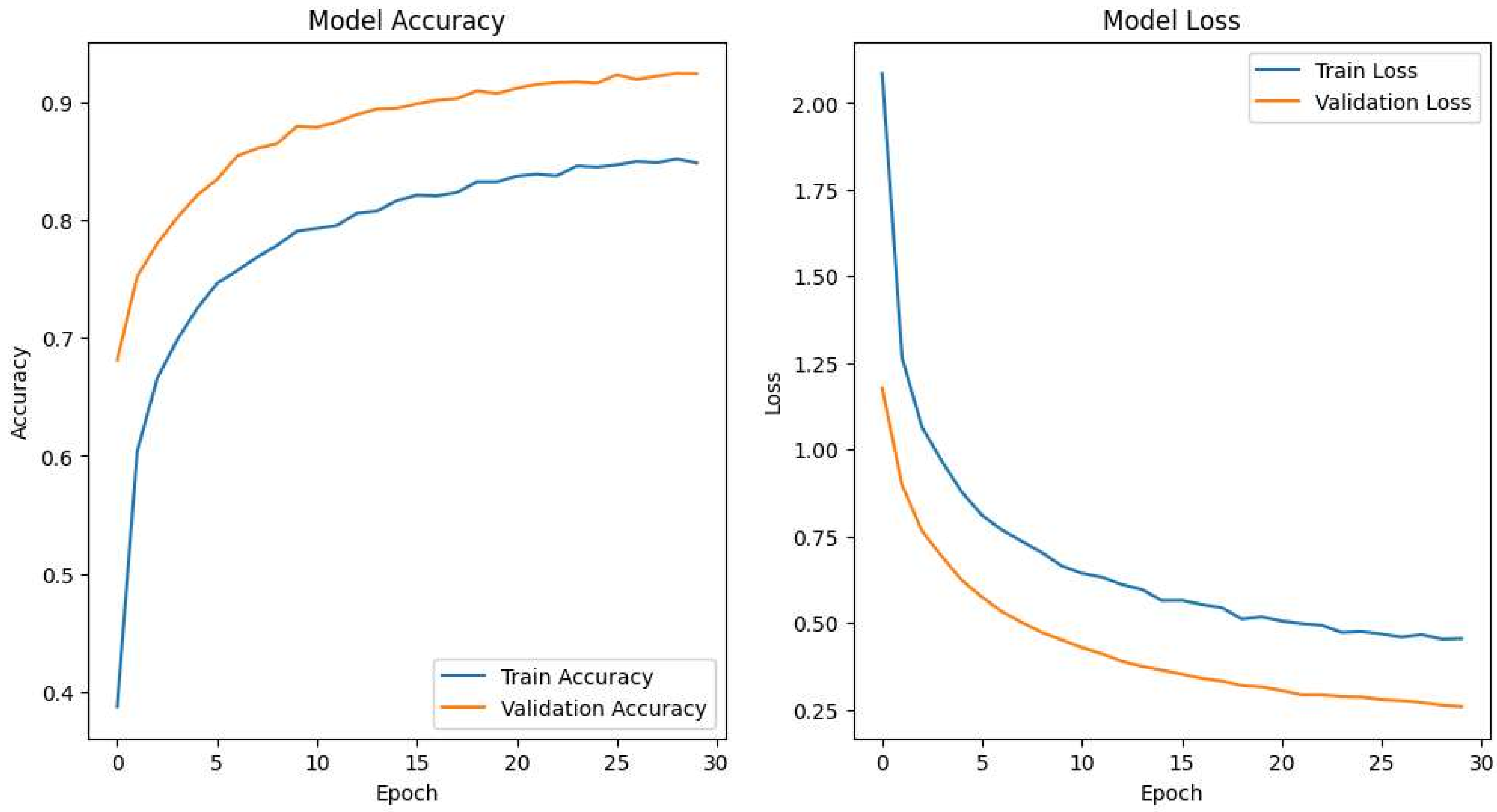
```
import matplotlib.pyplot as plt

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.show()
```


Model Accuracy
Accuracy
Epoch
Train Accuracy
Validation Accuracy
Model Loss
Loss
Epoch
Train Loss
Validation Loss

```
In [ ]:
```