Digit Recognizer

Introduction

In this notebook, we will build and train a Convolutional Neural Network (CNN) for the task of handwritten digit recognition using the famous MNIST dataset. The goal is to achieve high accuracy in classifying handwritten digits from 0 to 9.

We will go through various steps of the machine learning pipeline, including data loading, data visualization, data preprocessing, model building, training, evaluation, and prediction.

Let's get started!

import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from PIL import Image
# sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score , confusion_matrix
# tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense , Flatten , Conv2D , MaxPooling2D
from tensorflow.keras.utils import plot_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Exploary Data Analysis (EDA)

In [2]:
```
# Load the data
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

In [3]:
```
# Size of the datasets
print(f'train data shape ==> {train_data.shape}')
print(f'test data shape ==> {test_data.shape}')
```

```
train data shape ==> (42000, 785)
test data shape ==> (28000, 784)
```

In [4]:
```
train_data.head()
```

Out[4]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [5]:
```
test_data.head()
```

Out[5]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 784 columns

In [6]:
```
train_data.columns
```

Out[6]:
```
Index(['label', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=785)
```

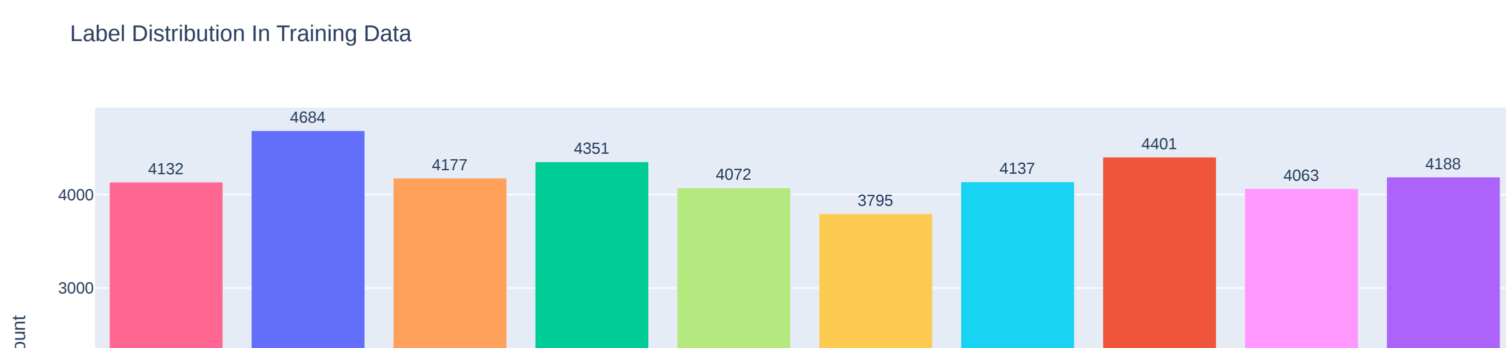
In [7]:
```
train_data.info()
```

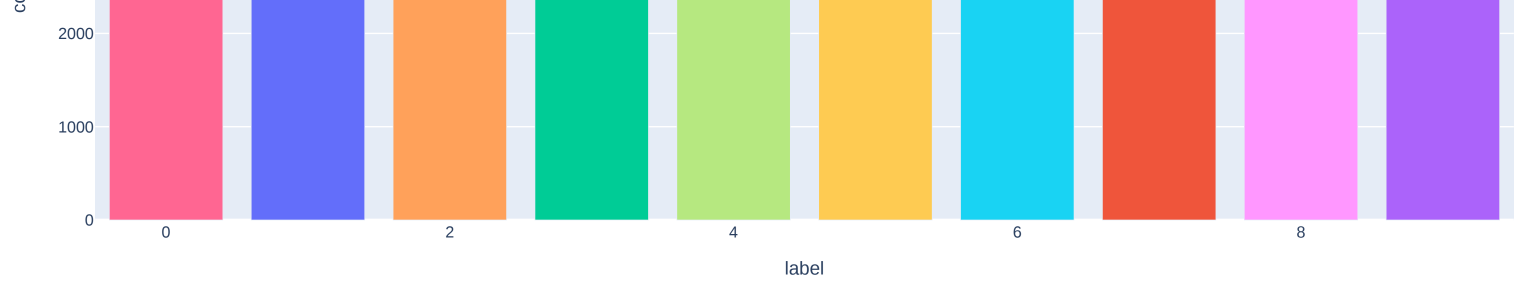
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

In [8]:
```
# Define the label counts
label_counts = train_data['label'].value_counts()
print(label_counts)
```

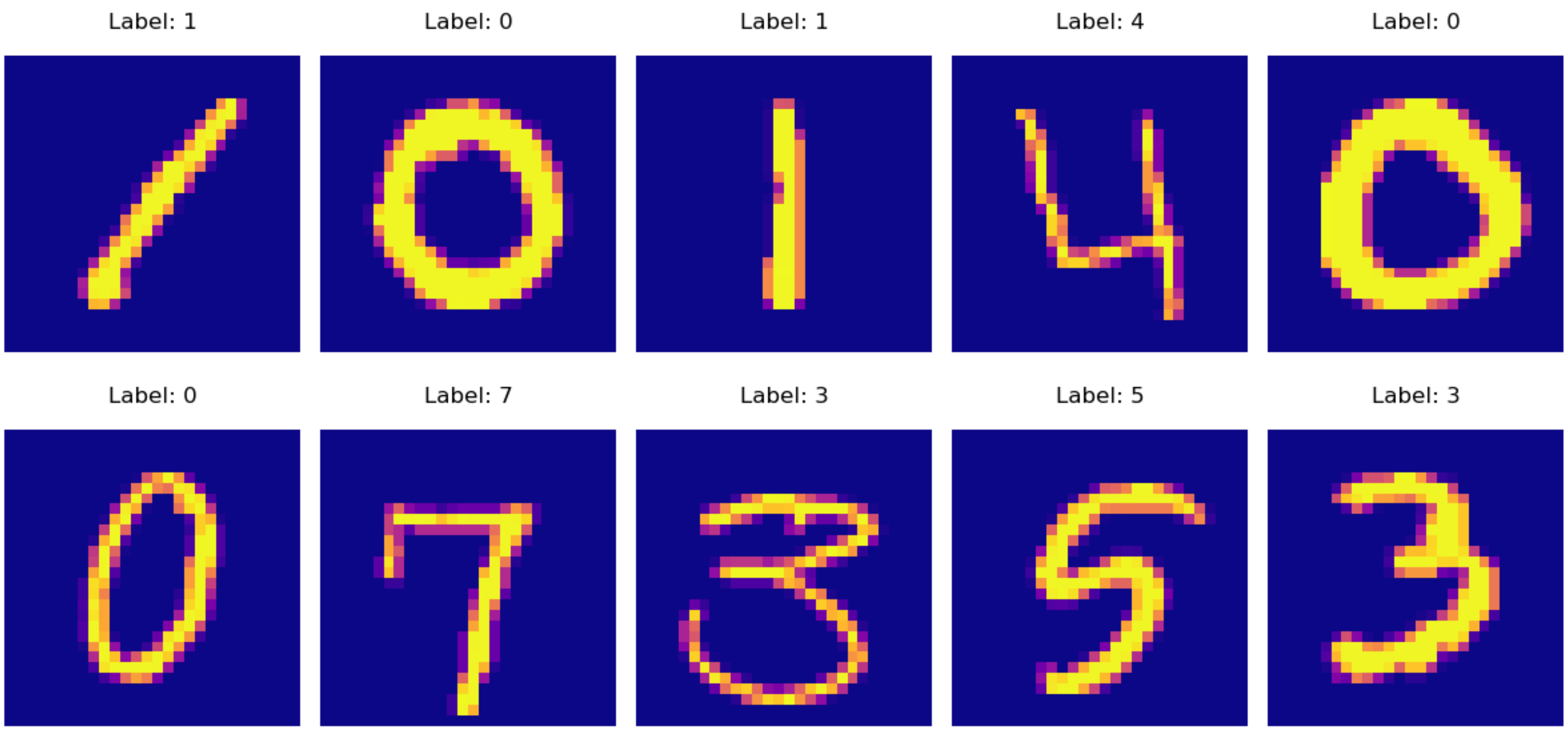
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

In [9]:
```
fig = px.bar(x=label_counts.index , y=label_counts.values ,labels = {'x':'label','y':'count'},
            text = label_counts.values ,title='Label Distribution In Training Data')
fig.update_traces(texttemplate='%{text}',textposition='outside',marker_color=px.colors.qualitative.Plotly)
fig.show()
```





In [10]:
```
# Visualize some digits
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(train_data.iloc[i, 1:].values.reshape(28, 28), cmap='plasma')
    plt.title(f"Label: {train_data.iloc[i, 0]}", pad=15)
    plt.axis('off')

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



Data Preprocessing

In [11]:
```
# Split the data into features and labels
X = train_data.drop('label', axis=1).values.astype('float32')
y = train_data['label'].values

# Normalize the pixel values to [0, 1]
```

```python
X /= 255.0

# Reshape the data to 28x28x1 (height, width, channels)
X = X.reshape(-1, 28, 28, 1)

# Convert labels to one-hot encoded vectors
y = tf.keras.utils.to_categorical(y, num_classes=10)

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [12]:

```python
print(f"X_train shape ==> {X_train.shape}")
print(f"X_val shape   ==> {X_val.shape}")
print(f"y_train shape ==> {y_train.shape}")
print(f"y_val shape   ==> {y_val.shape}")
```

```
X_train shape ==> (33600, 28, 28, 1)
X_val shape   ==> (8400, 28, 28, 1)
y_train shape ==> (33600, 10)
y_val shape   ==> (8400, 10)
```

Building and Training the Model (CNN)

In [45]:

```python
# Create the CNN model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Data Augmentation

In [46]:

```python
# Create a data generator with random transformations
datagen = ImageDataGenerator(rotation_range=10, width_shift_range=0.1, height_shift_range=0.1,
                             zoom_range=0.1, horizontal_flip=False, fill_mode='nearest')

# Fit the data generator on the training data
datagen.fit(X_train)

# Use the data generator during training
history = model.fit(datagen.flow(X_train, y_train, batch_size=128), epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
263/263 [==============================] - 20s 74ms/step - loss: 0.5369 - accuracy: 0.8324 - val_loss: 0.1153 - val_accuracy: 0.9639
Epoch 2/10
263/263 [==============================] - 19s 73ms/step - loss: 0.1881 - accuracy: 0.9436 - val_loss: 0.1027 - val_accuracy: 0.9685
Epoch 3/10
263/263 [==============================] - 20s 74ms/step - loss: 0.1298 - accuracy: 0.9606 - val_loss: 0.0579 - val_accuracy: 0.9826
Epoch 4/10
263/263 [==============================] - 19s 72ms/step - loss: 0.1040 - accuracy: 0.9682 - val_loss: 0.0497 - val_accuracy: 0.9840
Epoch 5/10
263/263 [==============================] - 19s 74ms/step - loss: 0.0911 - accuracy: 0.9725 - val_loss: 0.0495 - val_accuracy: 0.9835
Epoch 6/10
263/263 [==============================] - 19s 72ms/step - loss: 0.0790 - accuracy: 0.9747 - val_loss: 0.0490 - val_accuracy: 0.9849
Epoch 7/10
263/263 [==============================] - 19s 71ms/step - loss: 0.0707 - accuracy: 0.9774 - val_loss: 0.0375 - val_accuracy: 0.9874
Epoch 8/10
263/263 [==============================] - 20s 75ms/step - loss: 0.0652 - accuracy: 0.9803 - val_loss: 0.0353 - val_accuracy: 0.9889
Epoch 9/10
```

```
263/263 [==============================] - 20s 76ms/step - loss: 0.0615 - accuracy: 0.9804 - val_loss: 0.0392 - val_accuracy: 0.9879
Epoch 10/10
263/263 [==============================] - 20s 74ms/step - loss: 0.0587 - accuracy: 0.9817 - val_loss: 0.0355 - val_accuracy: 0.9890
```

Learning Rate Scheduling

In [15]:

```python
# Define a learning rate schedule function
def lr_schedule(epoch):
    initial_lr = 0.001
    if epoch < 5:
        return initial_lr
    else:
        return initial_lr * tf.math.exp(0.1 * (5 - epoch))

# Use the learning rate schedule during training
lr_scheduler = LearningRateScheduler(lr_schedule)
history = model.fit(X_train, y_train, batch_size=128, epochs=10, validation_data=(X_val, y_val), callbacks=[lr_scheduler])
```

```
Epoch 1/10
263/263 [==============================] - 38s 143ms/step - loss: 0.0263 - accuracy: 0.9920 - val_loss: 0.0349 - val_accuracy: 0.9887 - lr: 0.0010
Epoch 2/10
263/263 [==============================] - 38s 146ms/step - loss: 0.0174 - accuracy: 0.9945 - val_loss: 0.0250 - val_accuracy: 0.9919 - lr: 0.0010
Epoch 3/10
263/263 [==============================] - 43s 165ms/step - loss: 0.0130 - accuracy: 0.9960 - val_loss: 0.0239 - val_accuracy: 0.9919 - lr: 0.0010
Epoch 4/10
263/263 [==============================] - 41s 157ms/step - loss: 0.0107 - accuracy: 0.9970 - val_loss: 0.0258 - val_accuracy: 0.9915 - lr: 0.0010
Epoch 5/10
263/263 [==============================] - 44s 168ms/step - loss: 0.0077 - accuracy: 0.9977 - val_loss: 0.0278 - val_accuracy: 0.9918 - lr: 0.0010
Epoch 6/10
263/263 [==============================] - 38s 143ms/step - loss: 0.0065 - accuracy: 0.9981 - val_loss: 0.0318 - val_accuracy: 0.9898 - lr: 0.0010
Epoch 7/10
263/263 [==============================] - 26s 97ms/step - loss: 0.0057 - accuracy: 0.9981 - val_loss: 0.0284 - val_accuracy: 0.9918 - lr: 9.0484e-04
Epoch 8/10
263/263 [==============================] - 13s 49ms/step - loss: 0.0033 - accuracy: 0.9992 - val_loss: 0.0248 - val_accuracy: 0.9923 - lr: 8.1873e-04
Epoch 9/10
263/263 [==============================] - 13s 50ms/step - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.0266 - val_accuracy: 0.9930 - lr: 7.4082e-04
Epoch 10/10
263/263 [==============================] - 13s 50ms/step - loss: 0.0016 - accuracy: 0.9997 - val_loss: 0.0240 - val_accuracy: 0.9929 - lr: 6.7032e-04
```

Regularization (Dropout)

In [16]:

```python
# Create the CNN model with dropout layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Add dropout here
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Early Stopping

In [17]:

```python
# Use early stopping during training
early_stopping = EarlyStopping(patience=3, restore_best_weights=True)
history = model.fit(X_train, y_train, batch_size=128, epochs=50, validation_data=(X_val, y_val), callbacks=[early_stopping])
```

```
Epoch 1/50
263/263 [==============================] - 14s 49ms/step - loss: 0.4441 - accuracy: 0.8630 - val_loss: 0.1055 - val_accuracy: 0.9689
Epoch 2/50
263/263 [==============================] - 14s 52ms/step - loss: 0.1264 - accuracy: 0.9628 - val_loss: 0.0619 - val_accuracy: 0.9808
Epoch 3/50
263/263 [==============================] - 13s 51ms/step - loss: 0.0969 - accuracy: 0.9704 - val_loss: 0.0600 - val_accuracy: 0.9800
Epoch 4/50
263/263 [==============================] - 13s 50ms/step - loss: 0.0746 - accuracy: 0.9772 - val_loss: 0.0473 - val_accuracy: 0.9858
Epoch 5/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0668 - accuracy: 0.9804 - val_loss: 0.0440 - val_accuracy: 0.9861
Epoch 6/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0534 - accuracy: 0.9837 - val_loss: 0.0410 - val_accuracy: 0.9877
Epoch 7/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0498 - accuracy: 0.9846 - val_loss: 0.0399 - val_accuracy: 0.9871
Epoch 8/50
263/263 [==============================] - 13s 48ms/step - loss: 0.0430 - accuracy: 0.9868 - val_loss: 0.0338 - val_accuracy: 0.9901
Epoch 9/50
263/263 [==============================] - 13s 48ms/step - loss: 0.0398 - accuracy: 0.9878 - val_loss: 0.0356 - val_accuracy: 0.9892
Epoch 10/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0358 - accuracy: 0.9876 - val_loss: 0.0365 - val_accuracy: 0.9892
Epoch 11/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0339 - accuracy: 0.9888 - val_loss: 0.0307 - val_accuracy: 0.9901
Epoch 12/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0285 - accuracy: 0.9908 - val_loss: 0.0314 - val_accuracy: 0.9902
Epoch 13/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0290 - accuracy: 0.9909 - val_loss: 0.0329 - val_accuracy: 0.9905
Epoch 14/50
263/263 [==============================] - 12s 47ms/step - loss: 0.0264 - accuracy: 0.9910 - val_loss: 0.0310 - val_accuracy: 0.9905
```

Model Evaluation

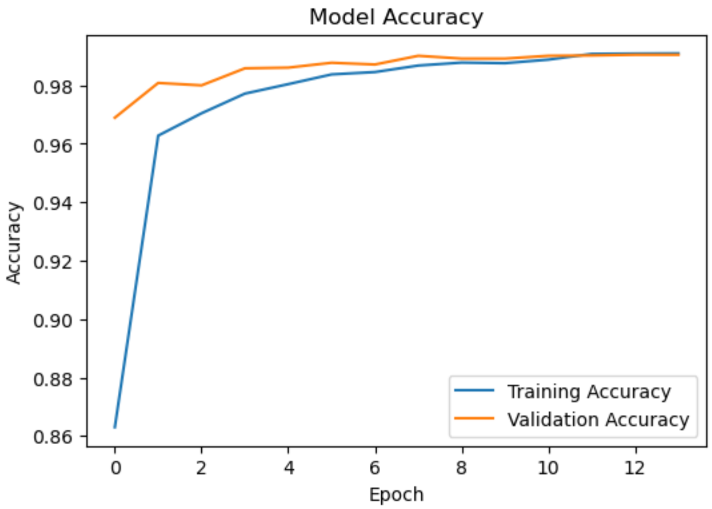
In [18]:
```python
# Smmary of the model
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dense_2 (Dense)             (None, 128)               204928

 dropout (Dropout)           (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 225034 (879.04 KB)
Trainable params: 225034 (879.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [19]:
```python
# Evaluate the model
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy:.4f}")
```

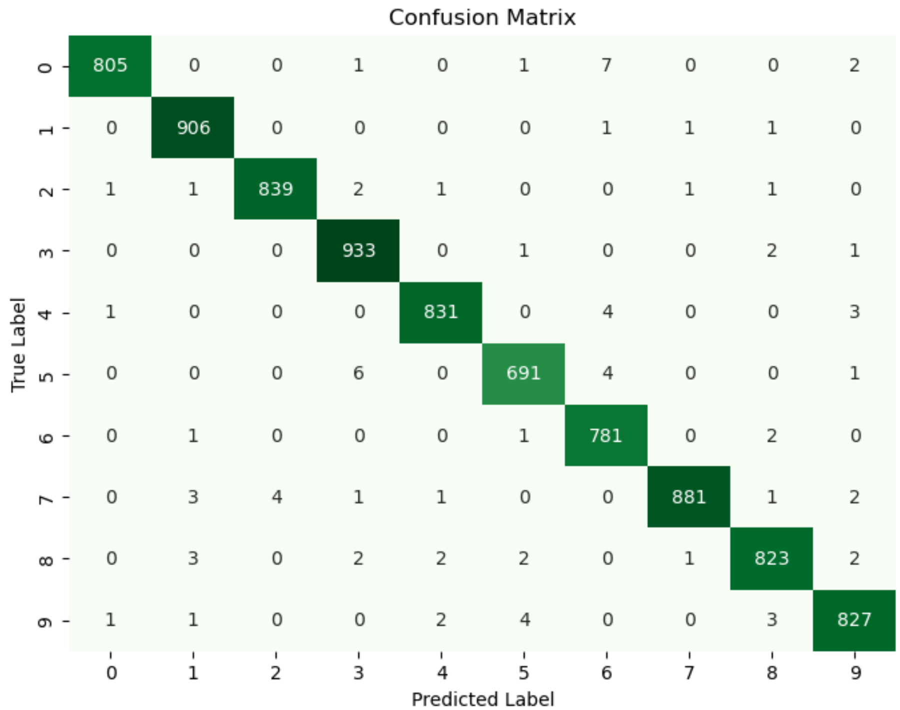
```
263/263 [==============================] - 1s 5ms/step - loss: 0.0307 - accuracy: 0.9901
Validation Accuracy: 0.9901
```

In [20]:
```
# Plot learning curves
plt.figure(figsize=(6, 4))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [21]:
```
# Create the confusion matrix
y_pred_val = np.argmax(model.predict(X_val), axis=1)
cm = confusion_matrix(np.argmax(y_val, axis=1), y_pred_val)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
263/263 [==============================] - 1s 5ms/step
```



Misclassified Examples

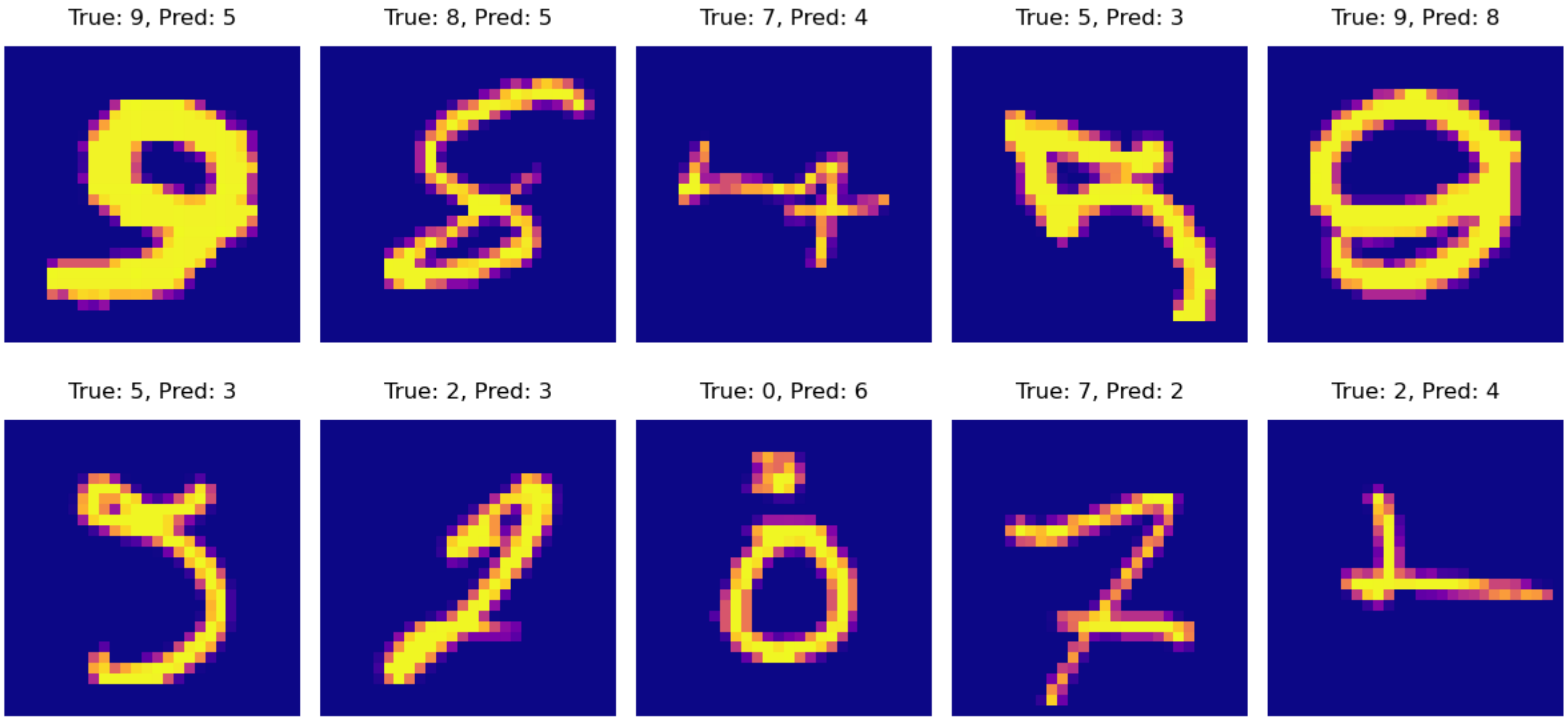
In [22]:

```
# Find misclassified examples
misclassified_idx = np.where(y_pred_val != np.argmax(y_val, axis=1))[0]

# Count the number of misclassified examples
num_misclassified = len(misclassified_idx)

# Print the count
print(f"Number of Misclassified Examples: {num_misclassified}")
```

```
Number of Misclassified Examples: 83
```

In [58]:

```
# Plot some misclassified examples
plt.figure(figsize=(12, 6))
for i, idx in enumerate(misclassified_idx[:10]):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_val[idx].reshape(28, 28), cmap='plasma')
    plt.title(f"True: {np.argmax(y_val[idx])}, Pred: {y_pred_val[idx]}", pad=12)
    plt.axis('off')
plt.tight_layout()
plt.show()
```



Making Predictions and Generating Submission File

In [47]:
```
# Preprocess test data
X_test = test_data.values.astype('float32')
X_test /= 255.0
X_test = X_test.reshape(-1, 28, 28, 1)

# Make predictions
predictions = model.predict(X_test)
y_pred = np.argmax(predictions, axis=1)
```

```
875/875 [==============================] - 4s 5ms/step
```

In [25]:
```
print(f"X_test shape ==> {X_test.shape}")
print(f"y_pred shape ==> {y_pred.shape}")
```

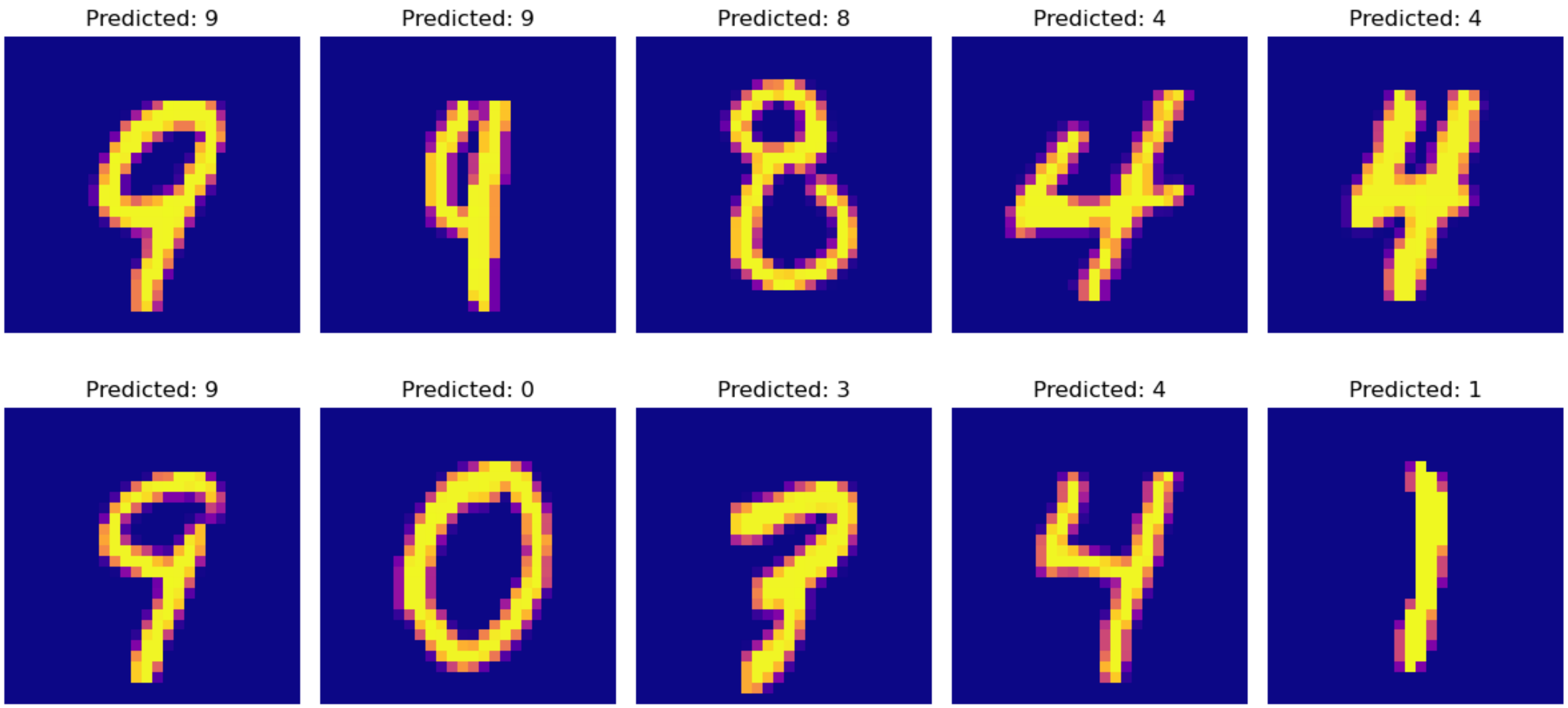
```
X_test shape ==> (28000, 28, 28, 1)
y_pred shape ==> (28000,)
```

In [26]:
```
# Save the entire model to a file
model.save('trained_model.keras')
```

Displaying Some Predicted Images

In [59]:
```
# Randomly select a few examples from the test set
num_examples_to_display = 10
random_indices = np.random.choice(len(X_test), num_examples_to_display, replace=False)
selected_images = X_test[random_indices]
selected_labels_true = y_pred[random_indices]
```

```python
# Display the selected images along with their predicted labels
plt.figure(figsize=(12, 6))
for i in range(num_examples_to_display):
    plt.subplot(2, 5, i + 1)
    plt.imshow(selected_images[i].reshape(28, 28), cmap='plasma')
    plt.title(f"Predicted: {selected_labels_true[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```



In [28]:

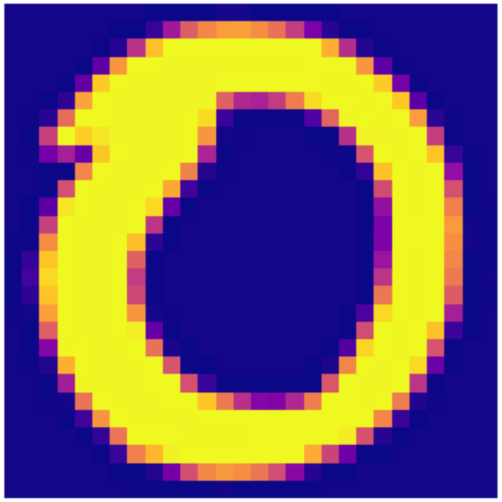
```python
# Create submission file
submission = pd.DataFrame({'ImageId': np.arange(1, len(y_pred)+1), 'Label': y_pred})
submission.to_csv('submission.csv', index=False)
```

In [60]:

```python
def predict_user_image(file_path):
    try:
        # Load and preprocess the user-provided image
        user_image = Image.open(file_path).convert('L')  # Convert to grayscale
        user_image = user_image.resize((28, 28))  # Resize to 28x28 pixels
        user_image = np.array(user_image)  # Convert to NumPy array

        # Invert pixel values to get black background and white number
        user_image = 255 - user_image

        user_image = user_image.astype('float32') / 255.0  # Normalize (assuming you used this preprocessing before)
        user_image = user_image.reshape(1, 28, 28, 1)  # Reshape to match the model's input shape

        # Make predictions using the trained model
        user_prediction = model.predict(user_image)
        user_label = np.argmax(user_prediction)

        # Display the user-provided image and the predicted label
        plt.imshow(user_image.reshape(28, 28), cmap='plasma')
        plt.title(f"Predicted: {user_label}")
        plt.axis('off')
        plt.show()

    except Exception as e:
        print("Error: ", e)
```

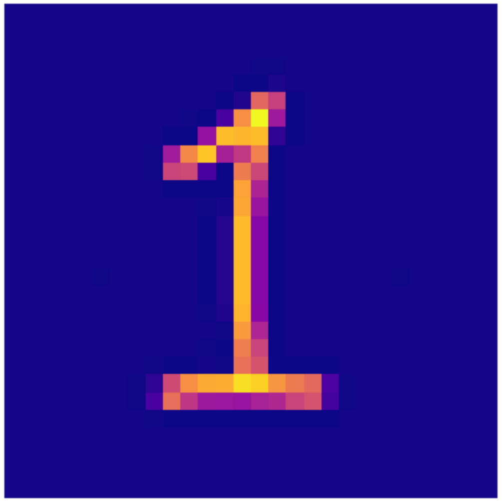
In [61]:
```
predict_user_image("0.jpg")
```
```
1/1 [==============================] - 0s 19ms/step
```

Predicted: 0



In [62]:
```
predict_user_image("1.jpg")
```
```
1/1 [==============================] - 0s 20ms/step
```

Predicted: 1



In [63]:
```
predict_user_image("2.jpg")
```
```
1/1 [==============================] - 0s 20ms/step
```

Predicted: 2

In [64]:
```
predict_user_image("3.jpg")
```

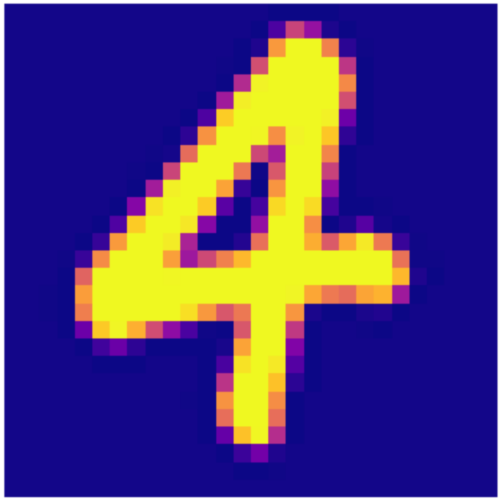
```
1/1 [==============================] - 0s 19ms/step
```

Predicted: 3

In [65]:
```
predict_user_image("4.jpg")
```

```
1/1 [==============================] - 0s 19ms/step
```

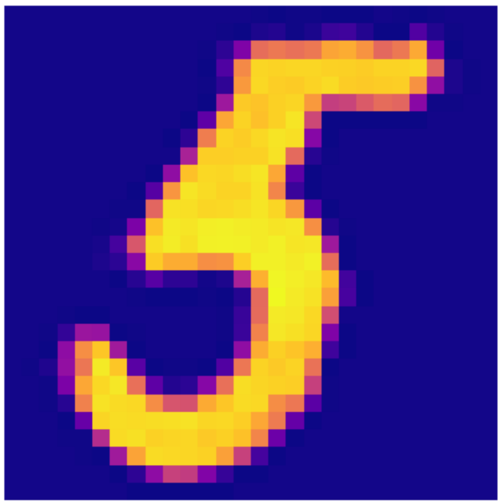
Predicted: 4



In [66]:
```
predict_user_image("5.jpg")
```

```
1/1 [==============================] - 0s 20ms/step
```

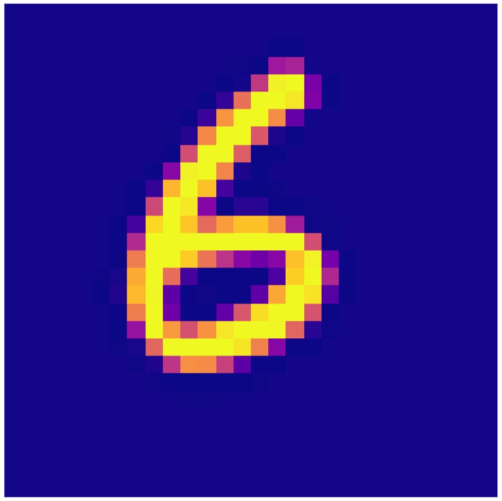
Predicted: 5



In [67]:
```
predict_user_image("6.jpg")
```

```
1/1 [==============================] - 0s 20ms/step
```

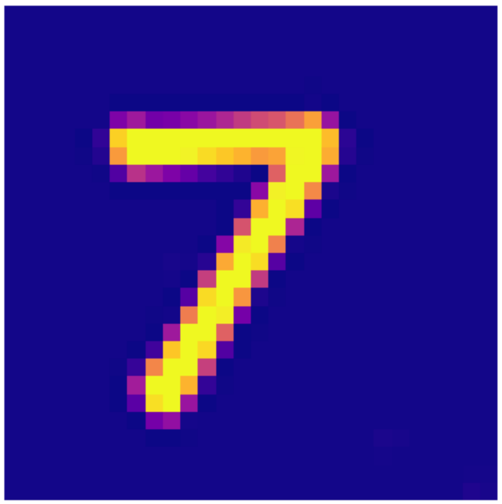
Predicted: 6



In [68]:
```
predict_user_image("7.jpg")
```

```
1/1 [==============================] - 0s 18ms/step
```

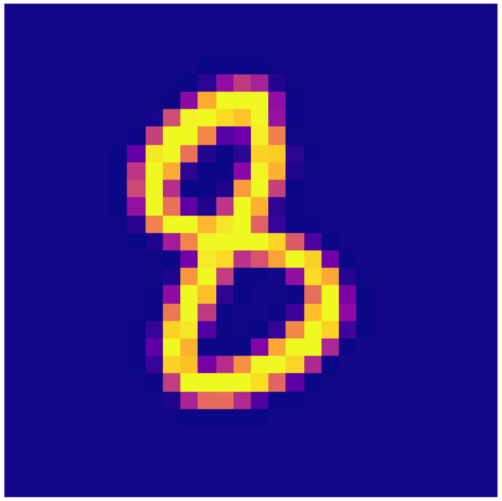
Predicted: 7



In [69]:
```
predict_user_image("8.jpg")
```

```
1/1 [==============================] - 0s 19ms/step
```

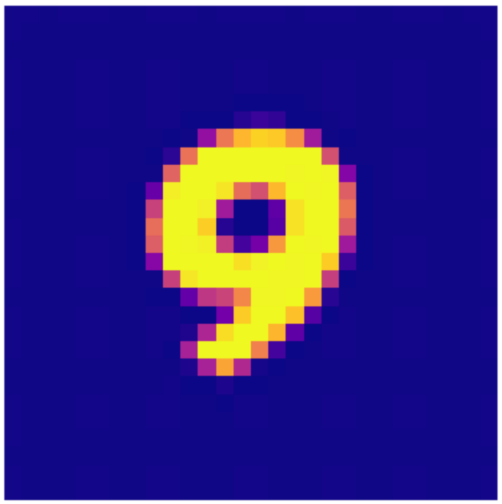
Predicted: 8



In [70]:
```
predict_user_image("9.jpg")
```

```
1/1 [==============================] - 0s 19ms/step
```

Predicted: 9



Conclusion

- In this project, we successfully built and trained a CNN model for handwritten digit recognition. We performed data augmentation, implemented learning rate scheduling, applied dropout regularization, and used early stopping to prevent overfitting.
- Our trained model achieved impressive accuracy on the validation set and was able to accurately predict digits from user-provided images as well.
- We also analyzed misclassified examples and visualized the model's performance using confusion matrices and learning curves.
- Overall, this project demonstrates the power of deep learning and CNNs in solving image classification tasks.

Thank you for following along and I hope to upvote it.

Made by: **Ahmed Sheta**

In [ ]:

```

```