In [5]:

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Input, BatchNormaliz
from keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
from PIL import Image, ImageDraw
import tkinter as tk
from tkinter import Button, Canvas, messagebox

# Load the MNIST dataset
def load_large_dataset():
    (X_train, y_train), (_, _) = mnist.load_data()
    return X_train, y_train

X_train, y_train = load_large_dataset()

# Convert class vectors to binary class matrices (one-hot encoding)
y_train = to_categorical(y_train, 10)

# Reshape to be samples*pixels*width*height
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')

# Normalize input data
X_train /= 255

# Create the CNN model
def create_model():
    model = Sequential()
    model.add(Input(shape=(28, 28, 1)))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    return model

# Compile and fit the model
model = create_model()
model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
model.fit(X_train, y_train, epochs=25, batch_size=128)

# Function to preprocess the drawn image for prediction
def preprocess_image(image):
    image = image.convert('L')  # Convert to grayscale
    image = image.resize((28, 28), Image.Resampling.LANCZOS)
    image_array = np.array(image)
    image_array = 255 - image_array  # Invert colors
    image_array = image_array.astype('float32') / 255.0
    image_array = image_array.reshape(1, 28, 28, 1)
```

```
    return image_array

# Function to predict the digit drawn on the canvas
def predict(image):
    preprocessed_image = preprocess_image(image)
    pred = model.predict(preprocessed_image, batch_size=1)
    return pred.argmax()

# Tkinter application for drawing and predicting digits
class DrawingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Digit Drawing and Prediction")

        # Create canvas
        self.canvas = Canvas(root, width=280, height=280, bg='white')
        self.canvas.pack()

        # Add buttons
        self.button_clear = Button(root, text="Clear", command=self.clear)
        self.button_clear.pack(side='left')

        self.button_predict = Button(root, text="Predict", command=self.predict)
        self.button_predict.pack(side='left')

        # Initialize drawing variables
        self.drawing = False
        self.last_x, self.last_y = 0, 0

        # Bind canvas events
        self.canvas.bind("<Button-1>", self.start_drawing)
        self.canvas.bind("<B1-Motion>", self.draw)

        # Create image for prediction
        self.image = Image.new("L", (280, 280), color=255)
        self.draw_image = ImageDraw.Draw(self.image)

    def start_drawing(self, event):
        self.drawing = True
        self.last_x, self.last_y = event.x, event.y

    def draw(self, event):
        if self.drawing:
            x, y = event.x, event.y
            self.canvas.create_line((self.last_x, self.last_y, x, y), fill='black',
            self.draw_image.line([self.last_x, self.last_y, x, y], fill=0, width=8)
            self.last_x, self.last_y = x, y

    def clear(self):
        self.canvas.delete("all")
        self.image = Image.new("L", (280, 280), color=255)
        self.draw_image = ImageDraw.Draw(self.image)

    def predict(self):
        prediction = predict(self.image)
        messagebox.showinfo("Prediction", f"The model predicts: {prediction}")
```

```
# Run the application
if __name__ == "__main__":
    root = tk.Tk()
    app = DrawingApp(root)
    root.mainloop()
```

```
Epoch 1/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 89ms/step - accuracy: 0.9178 - loss: 0.3582
Epoch 2/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 92ms/step - accuracy: 0.9891 - loss: 0.0348
Epoch 3/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 41s 88ms/step - accuracy: 0.9935 - loss: 0.0229
Epoch 4/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 96ms/step - accuracy: 0.9955 - loss: 0.0131
Epoch 5/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 92ms/step - accuracy: 0.9959 - loss: 0.0105
Epoch 6/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 94ms/step - accuracy: 0.9953 - loss: 0.0135
Epoch 7/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 48s 103ms/step - accuracy: 0.9953 - loss: 0.0167
Epoch 8/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 99ms/step - accuracy: 0.9970 - loss: 0.0098
Epoch 9/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 94ms/step - accuracy: 0.9974 - loss: 0.0080
Epoch 10/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 91ms/step - accuracy: 0.9971 - loss: 0.0095
Epoch 11/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 42s 89ms/step - accuracy: 0.9975 - loss: 0.0085
Epoch 12/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 46s 99ms/step - accuracy: 0.9977 - loss: 0.0074
Epoch 13/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 93ms/step - accuracy: 0.9978 - loss: 0.0099
Epoch 14/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 46s 98ms/step - accuracy: 0.9984 - loss: 0.0054
Epoch 15/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 48s 102ms/step - accuracy: 0.9985 - loss: 0.0062
Epoch 16/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 95ms/step - accuracy: 0.9982 - loss: 0.0069
Epoch 17/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 96ms/step - accuracy: 0.9991 - loss: 0.0033
Epoch 18/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 48s 103ms/step - accuracy: 0.9987 - loss: 0.0043
Epoch 19/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 46s 98ms/step - accuracy: 0.9981 - loss: 0.0074
Epoch 20/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 101ms/step - accuracy: 0.9977 - loss: 0.0103
Epoch 21/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 95ms/step - accuracy: 0.9988 - loss: 0.0047
Epoch 22/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 46s 97ms/step - accuracy: 0.9993 - loss: 0.0022
Epoch 23/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 91ms/step - accuracy: 0.9997 - loss: 0.0010
Epoch 24/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 95ms/step - accuracy: 0.9993 - loss: 0.0025
Epoch 25/25
469/469 ━━━━━━━━━━━━━━━━━━━━ 47s 100ms/step - accuracy: 0.9981 - loss: 0.0083
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

In [ ]:

Detailed Report on "Digit Drawing and Prediction" Application Using a Convolutional Neural Network (CNN)

1. Introduction

This project demonstrates the creation of a deep learning model for digit recognition using a Convolutional Neural Network (CNN). The application allows users to draw a digit on a canvas, and the trained model predicts the digit based on the drawn image. This report details the steps involved in data preparation, model creation, training, and integration with a graphical user interface (GUI) built using Tkinter.

2. Dataset and Preprocessing

2.1 Dataset The project uses the MNIST dataset, which consists of 60,000 training images and 10,000 testing images. Each image represents a handwritten digit (0-9) and is 28x28 pixels in grayscale.

Training set: 60,000 images Test set: 10,000 images 2.2 Preprocessing Steps Reshaping: The images are reshaped to a 4D tensor with dimensions (samples, width, height, channels). For MNIST, this translates to (samples, 28, 28, 1), where 1 is the number of channels (grayscale).

Normalization: The pixel values, originally in the range [0, 255], are normalized to [0, 1] by dividing by 255.0.

One-Hot Encoding: The target labels are converted into one-hot encoded vectors using to_categorical, which is essential for multi-class classification.

3. Model Architecture

The CNN model is constructed using Keras with the following architecture:

Input Layer:

Shape: (28, 28, 1) (grayscale image of size 28x28) Convolutional Layers:

Conv2D (32 filters, 3x3 kernel, ReLU activation): Extracts features from the input image. MaxPooling2D (2x2): Reduces the dimensionality by downsampling the feature maps. Conv2D (64 filters, 3x3 kernel, ReLU activation): Extracts more complex features. Conv2D (64 filters, 3x3 kernel, ReLU activation): Further feature extraction. MaxPooling2D (2x2): Further downsampling. Fully Connected Layers:

Flatten: Converts the 2D matrix into a 1D vector. Dense (100 units, ReLU activation): Adds a fully connected layer with 100 units. Dense (10 units, Softmax activation): Output layer with 10 units (corresponding to digits 0-9). Optimizer:

SGD (Stochastic Gradient Descent): Used for optimization with default settings. Loss Function:

Categorical Crossentropy: Suitable for multi-class classification tasks.

4. Model Training and Evaluation

4.1 Training The model is trained using the training set with the following parameters:

Batch Size: 128 Epochs: 20 Validation Data: The test set is used to validate the model during training. Metrics: Accuracy is used to monitor the training process. The training process aims to minimize the categorical crossentropy loss, thereby increasing the accuracy of the model on unseen data.

4.2 Evaluation Post-training, the model is evaluated on the test set to determine its performance. The evaluation metrics include:

Test Loss: The loss value on the test set. Test Accuracy: The percentage of correctly predicted digits on the test set. python Copy code score = model.evaluate(X_test, y_test, verbose=0) print('Test loss:', score[0]) print('Test accuracy:', score[1]) These metrics provide insight into the model's generalization ability.

5. Model Serialization

To use the model for predictions in a real-time application, it is serialized into JSON format, and the weights are saved separately.

Model Architecture: Saved as model.json. Model Weights: Saved as final_model.weights.h5. This allows the model to be loaded later for predictions without needing to retrain.

6. Image Preprocessing for Prediction

The preprocessing steps for the images drawn on the canvas ensure that they are compatible with the trained model:

Grayscale Conversion: The drawn image is converted to grayscale. Resizing: The image is resized to 28x28 pixels to match the input shape expected by the model. Color Inversion: The colors are inverted because MNIST images are white digits on a black background. Normalization: The pixel values are normalized to the range [0, 1]. Reshaping: The image is reshaped to (1, 28, 28, 1) for batch prediction. 7. Prediction Functionality The prediction function loads the pre-trained model and uses it to predict the digit drawn on the canvas:

python Copy code def predict(image): preprocessed_image = preprocess_image(image) pred = model.predict(preprocessed_image, batch_size=1) return pred.argmax() The function returns the digit with the highest probability.

8. Graphical User Interface (GUI)

The GUI is built using the Tkinter library, providing an intuitive interface for users to draw digits and receive predictions.

8.1 Canvas and Drawing Canvas: A drawing area where users can draw digits with the mouse. Drawing: Users can draw lines on the canvas, and the drawing is captured as an image. 8.2 Buttons Clear Button: Clears the canvas for a new drawing. Predict Button: Triggers the prediction process and displays the predicted digit in a message box. python Copy code self.button_clear = Button(root, text="Clear", command=self.clear) self.button_predict = Button(root, text="Predict", command=self.predict)

9. Application Workflow

Launch Application: The application window opens with a blank canvas. Draw Digit: The user draws a digit using the mouse. Predict Digit: The user clicks the "Predict" button, and the application processes the image, predicts the digit, and displays the result. Clear Canvas: The user can clear the canvas and draw a new digit.

11. Conclusion

This project successfully integrates a CNN model for digit recognition with a user-friendly GUI application. The CNN model, trained on the MNIST dataset, accurately predicts handwritten digits drawn by the user. The project demonstrates the practical application of deep learning in a simple yet effective manner, highlighting the potential for developing more advanced AI-driven interfaces.

This report provides a comprehensive overview of the design, implementation, and functionality of the "Digit Drawing and Prediction" application, making it a valuable tool for learning and exploring the capabilities of deep learning models in real-world applications.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Assuming model history is available as 'history' and prediction accuracy for draw
# has been recorded in a list called 'drawn_digit_accuracy'

# Example data for plotting
epochs = range(1, 21)  # Example epochs range
training_accuracy = [0.85, 0.88, 0.90, 0.92, 0.93, 0.94, 0.95, 0.96, 0.96, 0.97, 0.
drawn_digit_accuracy = [0.80, 0.83, 0.85, 0.86, 0.87, 0.88, 0.89, 0.90, 0.91, 0.92,

# Plotting the model training accuracy over epochs
plt.figure(figsize=(14, 7))

# Subplot 1: Training Accuracy
plt.subplot(1, 2, 1)
plt.plot(epochs, training_accuracy, 'b', label='Training Accuracy')
plt.title('Model Training Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.0])
```

```
plt.legend()

# Subplot 2: Drawn Digit Prediction Accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, drawn_digit_accuracy, 'r', label='Drawn Digit Prediction Accuracy'
plt.title('Prediction Accuracy on Drawn Digits')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.0])
plt.legend()

# Final layout and display
plt.tight_layout()
plt.show()
```