In [4]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(x_train.shape, y_train.shape)
```

```
(60000, 28, 28) (60000,)
```

In [5]:
```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
num_classes = 10
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)#this converts numbers to binary, length max of binary
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32') # helps when dealing with sigmoid or weighted sum
x_test = x_test.astype('float32')
#normalizing data, converting into [0,1] range
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [6]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, BatchNormalization

batch_size = 128
epochs = 3

model = Sequential()

# First block: Convolution, Batch Normalization, Activation, Pooling
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())#Stabilizes mean and variance, for speedy running
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())# Also to standardize so that the affects of weights of 1 layer do not relay to
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))# this reduces chances of overfitting

# Second block
model.add(Conv2D(128, (3, 3), activation='relu'))#128 to detect complex features
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))# greater drop out since the model has grown complex

# Third block
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

# Flatten and Dense Layers
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

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

```
C:\Users\SMRC\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv
.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models
, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [7]:
```
hist = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test
print("The model has successfully trained")

model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

```
Epoch 1/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 224s 454ms/step - accuracy: 0.6462 - loss: 1.1647 - val_accuracy: 0.6687 - val_loss
: 1.4737
Epoch 2/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 196s 418ms/step - accuracy: 0.9682 - loss: 0.1145 - val_accuracy: 0.9897 - val_loss
: 0.0351
Epoch 3/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 1886s 4s/step - accuracy: 0.9789 - loss: 0.0776 - val_accuracy: 0.9924 - val_loss:
0.0269
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
The model has successfully trained
Saving the model as mnist.h5
```

In [ ]:
```
model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

In [ ]:
```
from keras.models import load_model
import tkinter as tk
from tkinter import *
import numpy as np
from PIL import ImageGrab, Image, ImageOps

# Load the trained model
model = load_model('mnist.h5')

def predict_digit(img):
    """
    Preprocesses the image and uses the trained model to predict the digit.

    Parameters:
    img (PIL.Image): The image captured from the canvas.

    Returns:
    tuple: (predicted_digit, confidence)
    """
    # Resize image to 28x28 pixels
    img = img.resize((28, 28), Image.Resampling.LANCZOS)
    # Convert to grayscale
    img = img.convert('L')
    # Invert colors (MNIST has white digits on a black background)
    img = ImageOps.invert(img)
    # Convert to numpy array
    img = np.array(img)

    # Center and normalize the image
    img = crop_and_center_image(img)

    # Normalize the pixel values
    img = img / 255.0
    # Reshape to match the input shape of the model
    img = img.reshape(1, 28, 28, 1)

    # Predict the class
    res = model.predict(img)[0]
    return np.argmax(res), max(res)

def crop_and_center_image(img):
    """
    Crops the image to remove empty space, centers the digit, and pads to fit 28x28.
    """
    # Crop non-zero areas
    rows = np.where(np.any(img > 0, axis=1))[0]
    cols = np.where(np.any(img > 0, axis=0))[0]
    if rows.size and cols.size:
        img = img[rows.min():rows.max() + 1, cols.min():cols.max() + 1]

    # Resize cropped content to 20x20 pixels
    img = Image.fromarray(img).resize((20, 20), Image.Resampling.LANCZOS)
    img = np.pad(img, ((4, 4), (4, 4)), mode='constant', constant_values=0)
    return np.array(img)

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("MNIST Digit Recognizer")
```

```
        self.geometry("600x350")
        self.resizable(False, False)
        self.configure(bg="white")

        self.x = self.y = 0

        # Create elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg="white", cursor="cross")
        self.label = tk.Label(self, text="Draw a digit", font=("Helvetica", 18), bg="white")
        self.classify_btn = tk.Button(self, text="Recognize", command=self.classify_handwriting)
        self.clear_btn = tk.Button(self, text="Clear", command=self.clear_all)

        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, padx=2)
        self.label.grid(row=0, column=1, pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.clear_btn.grid(row=1, column=0, pady=2, padx=2)

        # Bind drawing functionality
        self.canvas.bind("<B1-Motion>", self.draw_lines)

    def clear_all(self):
        """Clears the canvas and resets the label."""
        self.canvas.delete("all")
        self.label.configure(text="Draw a digit")

    def classify_handwriting(self):
        """Captures the canvas content and predicts the digit."""
        # Update the GUI to ensure all drawings are rendered
        self.update()

        # Get the canvas's absolute position
        x = self.canvas.winfo_rootx()
        y = self.canvas.winfo_rooty()
        x1 = x + self.canvas.winfo_width()
        y1 = y + self.canvas.winfo_height()

        # Capture the canvas content
        im = ImageGrab.grab(bbox=(x, y, x1, y1))

        # Save debug image (optional)
        im.save("debug_image.png")

        # Predict the digit
        digit, acc = predict_digit(im)
        if acc > 0.7:  # Increased confidence threshold
            self.label.configure(text=f"Digit: {digit}, {int(acc * 100)}%")
        else:
            self.label.configure(text="Uncertain Prediction")

    def draw_lines(self, event):
        """Draws a line on the canvas."""
        r = 3  # Smaller brush size for accuracy
        # Draw an oval (circle) to simulate brush strokes
        self.canvas.create_oval(event.x - r, event.y - r, event.x + r, event.y + r, fill='black', outline='blac

# Run the application
if __name__ == "__main__":
    app = App()
    app.mainloop()
```

In [ ]:

In [ ]:

In [ ]: