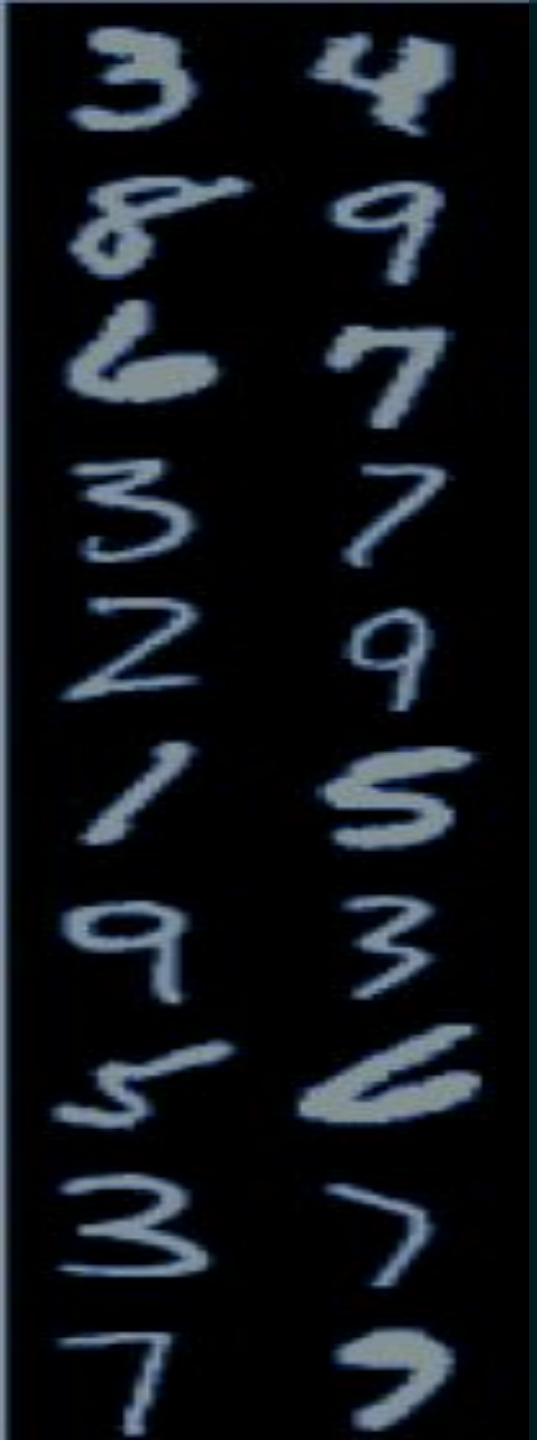


MNIST HANDWRITTEN DIGIT RECOGNITION

SQE Project

GROUP MEMBERS

- Aqsa Tabassum_02
- Shamsa Kanwal_028
- Sadia Azam_039

INTRODUCTION

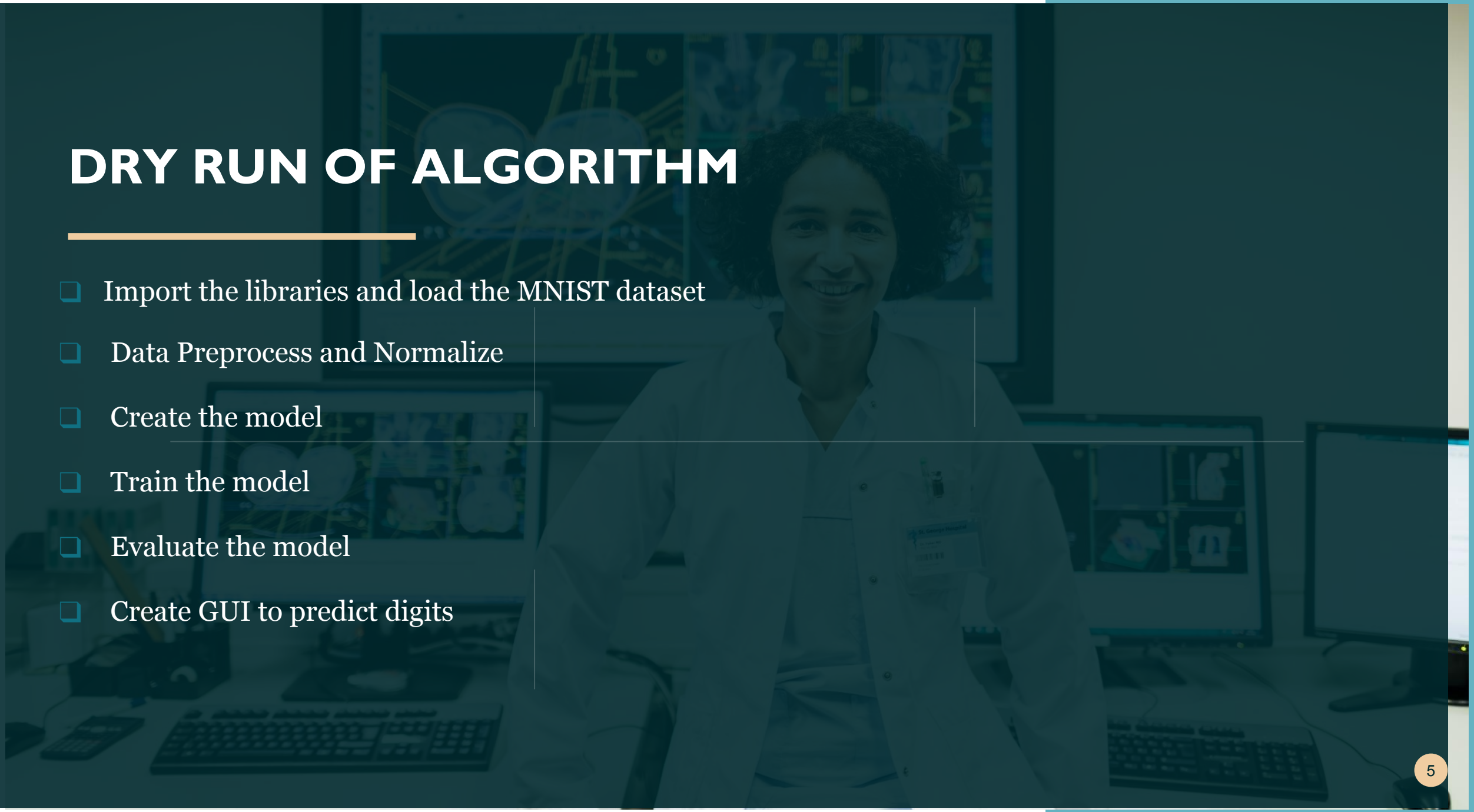
- Handwritten digits recognition could be a well-researched subarea.
- Involved with learning models to differentiate pre-segmented written digits.
- Main application of machine learning strategies has determined efficacious in conformist decisive systems.
- In 21st century written digit communication has its own normal
- Challenges in handwritten characters recognition lies within the variation and distortion of handwritten.
- The challenges are principally caused by the big variation of individual writing styles.
- Strong feature extraction is extremely important.
- Handwritten digit recognition has obtained heap of concentration.
- Character reognition system may function as a cornerstone.

WHICH ALGORITHM IS USED AND WHY?

- Neural Network: Training and testing wad done.
- CNN was found to have highest accuracy of 98%.
- It was used for the model.
- Both were tested for results in GUI user-based application.

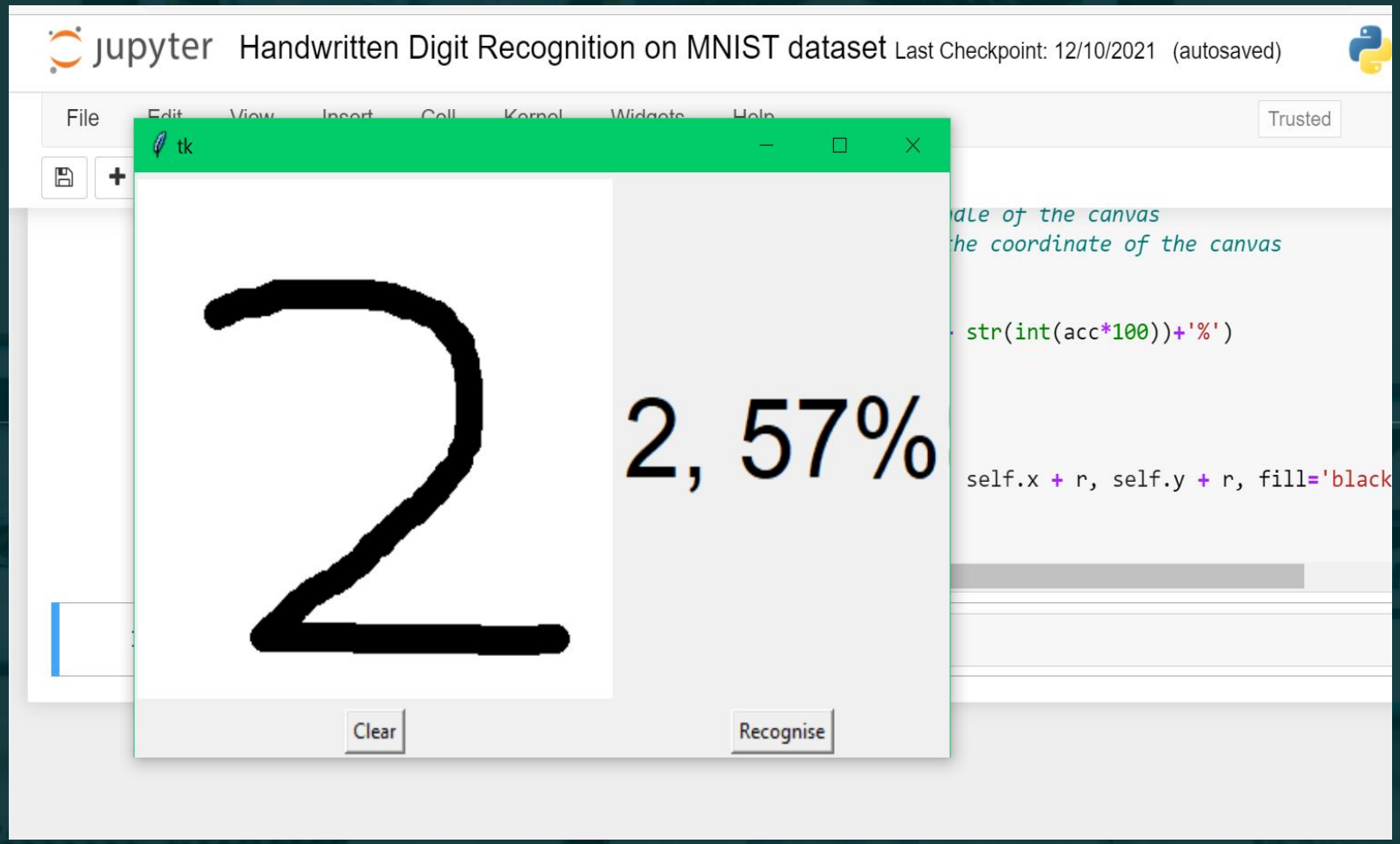
DRY RUN OF ALGORITHM

- Import the libraries and load the MNIST dataset
- Data Preprocess and Normalize
- Create the model
- Train the model
- Evaluate the model
- Create GUI to predict digits

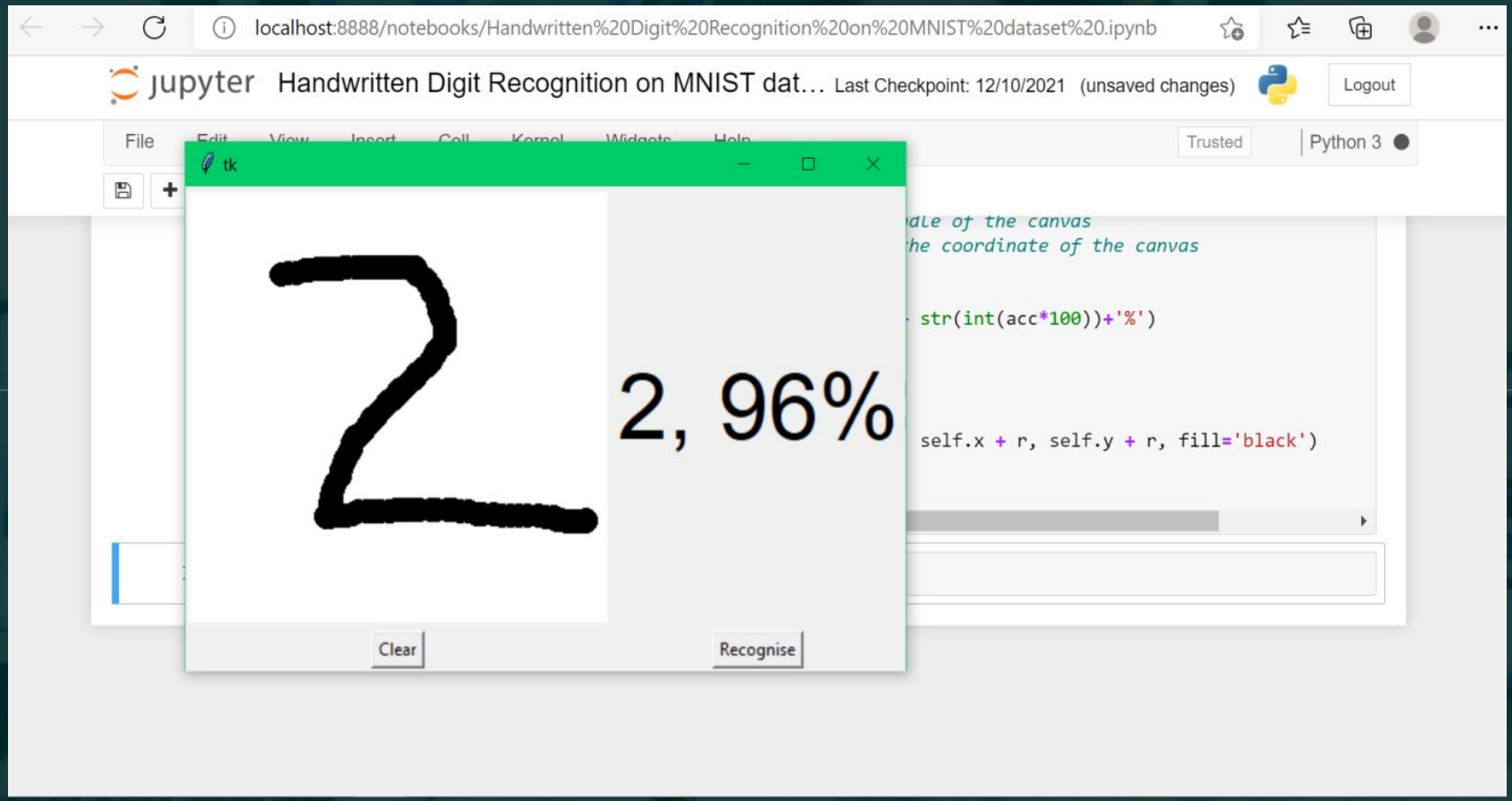


DRY RUN OF ALGORITHM

- .



DRY RUN OF ALGORITHM



TOOLS AND TECHNOLOGIES USED:

- We will be using a special type of deep neural network that is **Convolutional Neural Networks**. In the end, we are going to build a GUI in which you can draw the digit and recognize it straight away
- To achieve high performance on the handwritten digit recognition task using the **MNIST** dataset and build a **GUI App** based on **Tkinter.**

OUTPUT

Recognition Through Fully Covulutional Network Model.

In [6]:

```
#Building FCN Model
###4.Build the model
model = Sequential()
model.add(Dense(hidden1, input_dim=input_size, activation='relu'))
# output = relu (dot (W, input) + bias)
model.add(Dense(hidden2, activation='relu'))
model.add(Dense(classes, activation='softmax'))

# Compilation
model.compile(loss='categorical_crossentropy',
    metrics=['accuracy'], optimizer='sgd')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 400)               314000
_________________________________________________________________
dense_1 (Dense)              (None, 20)                8020
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
=================================================================
Total params: 322,230
Trainable params: 322,230
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
#Training the model
# Fitting on Data
model.fit(X_train, Y_train, batch_size=batch_size, epochs=10, verbose=2)
###5.Test
```

```
Epoch 1/10
300/300 - 5s - loss: 1.5074 - accuracy: 0.6038
Epoch 2/10
300/300 - 2s - loss: 0.7047 - accuracy: 0.8234
Epoch 3/10
300/300 - 2s - loss: 0.5072 - accuracy: 0.8679
Epoch 4/10
300/300 - 2s - loss: 0.4235 - accuracy: 0.8858
Epoch 5/10
300/300 - 2s - loss: 0.3772 - accuracy: 0.8948
Epoch 6/10
300/300 - 2s - loss: 0.3470 - accuracy: 0.9022
Epoch 7/10
300/300 - 2s - loss: 0.3251 - accuracy: 0.9081
Epoch 8/10
300/300 - 2s - loss: 0.3076 - accuracy: 0.9123
Epoch 9/10
300/300 - 2s - loss: 0.2931 - accuracy: 0.9164
Epoch 10/10
300/300 - 2s - loss: 0.2807 - accuracy: 0.9197
```

Out[7]: <keras.callbacks.History at 0x22592ec2e50>

OUTPUT

In [8]:

```
#Testing the model
score = model.evaluate(X_test, Y_test, verbose=1)
print('\n''Test accuracy:', score[1])
mask = range(10,20)
X_valid = X_test[mask]
y_pred = (model.predict(X_valid) > 0.5).astype("int32")
print(y_pred)
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(X_valid[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
plt.close()
model.save('C:\\Users\\Admin\\Downloads\\FCNN')
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2656 - accuracy: 0.9228

Test accuracy: 0.9228000044822693
[[1 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 1]
 [1 0 0 0 0 0 0 0 0 0]
 [0 1 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 1]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]]
```

```
INFO:tensorflow:Assets written to: C:\Users\Admin\Downloads\FCNN\assets
```

OUTPUT

Testing models through GUI Application

In [17]:

```
#GUI
from keras.models import load_model
from tkinter import *
import tkinter as tk
import win32gui
from PIL import ImageGrab, Image
import numpy as np
model = load_model('C:\\Users\\Admin\\Downloads\\CNN')
def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    #reshaping to support our model input and normalizing
    img = img.reshape(1,28,28,1)
    img = img/255.0
    #predicting the class
    res = model.predict([img])[0]
    return np.argmax(res), max(res)
class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self.x = self.y = 0
        # Creating elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")
        self.label = tk.Label(self, text="Thinking..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command =         self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Clear", command = self.clear_all)
        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)
        #self.canvas.bind("<Motion>", self.start_pos)
        self.canvas.bind("<B1-Motion>", self.draw_lines)
    def clear_all(self):
        self.canvas.delete("all")
    def classify_handwriting(self):
        HWND = self.canvas.winfo_id() # get the handle of the canvas
        rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas
        im = ImageGrab.grab(rect)
        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')
    def draw_lines(self, event):
        self.x = event.x
        self.y = event.y
        r=8
        self.canvas.create_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')
app = App()
mainloop()
```

THANK YOU