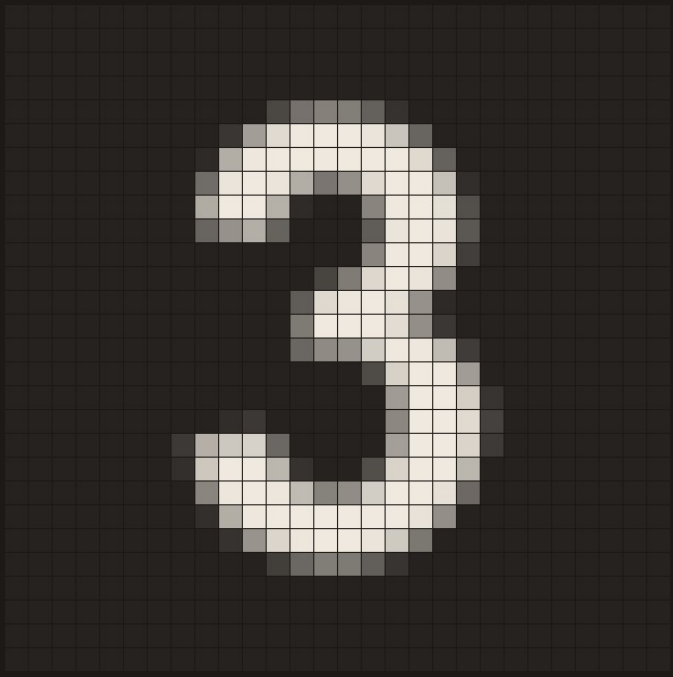
AI · COMPUTER VISION — PERSONAL PROJECT

Handwritten Digit Recognition

Reading numbers from a camera, in real time — with AI

A convolutional neural network trained on MNIST, deployed as a live web app that reads handwritten digits straight from your camera — and shows its confidence across all ten classes in real time.



Python	TensorFlow / Keras	OpenCV	NumPy	CNN	MNIST	Web App

Ömer Barış Berk

Founder & Developer

OVERVIEW

Teaching a browser to read handwriting

This is an end-to-end computer-vision project that recognises handwritten digits in real time using AI. A digit held up to the camera is isolated, cleaned up and passed through a convolutional neural network (CNN), which returns both the predicted number and a live probability distribution over the digits 0–9.

The aim was to go past a notebook experiment and ship a working product — handling the messy middle of camera input, image preprocessing and real-world lighting. It exists in two forms:

Desktop app · Python + OpenCV

Reads frames from a live webcam, isolates the digit in a region of interest, and shows the binarised image the model actually sees.

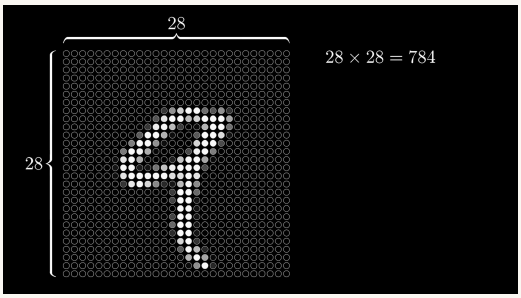
Web app · deployed

The same model running live in the browser, with a 0–9 confidence bar that makes every prediction transparent.

HOW IT WORKS

From camera frame to prediction

1. **Capture** OpenCV reads frames from the camera.
2. **Region of interest** A box isolates the digit from the background.
3. **Preprocess** Grayscale, resize to 28×28, normalise pixels to 0–1.
4. **Inference** The CNN predicts the digit via a softmax over 10 classes.
5. **Overlay** Prediction + confidence are drawn back on screen.



Each frame becomes a 28×28 grid — 784 input values for the network.

THE MODEL

A convolutional neural network, layer by layer

The data. The network is trained on MNIST — 70,000 grayscale images of handwritten digits (60,000 for training, 10,000 for testing), each 28×28 pixels. Pixel values are scaled to the 0–1 range so the model trains stably.

The architecture. Two convolutional blocks learn visual features — strokes and curves — then fully-connected layers turn those features into a digit. Max-pooling keeps the strongest signals and adds tolerance to small shifts; the final softmax outputs a probability for each of the ten classes.

Layer	Output shape
Input — grayscale image	28 × 28 × 1
Conv2D · 32 filters (3×3) · ReLU	26 × 26 × 32
MaxPooling2D (2×2)	13 × 13 × 32
Conv2D · 64 filters (3×3) · ReLU	11 × 11 × 64
MaxPooling2D (2×2)	5 × 5 × 64
Flatten	1600
Dense · 64 · ReLU	64
Dense · 10 · Softmax	10 (digits 0–9)

THE CODE

model.py

```
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax'), # one output per digit
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

Training. Optimised with Adam and cross-entropy loss, using a 90 / 10 train–validation split. Accuracy and loss curves were plotted with Matplotlib to confirm the model was learning and not over-fitting — it reaches roughly 98% accuracy on the MNIST test set.

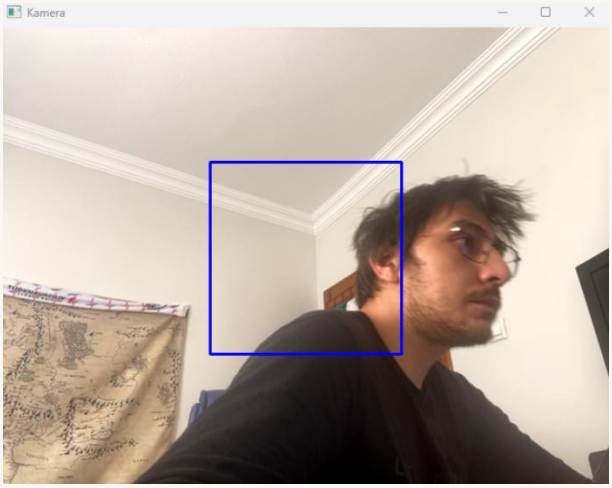
BUILDING IT

From webcam to a live recogniser

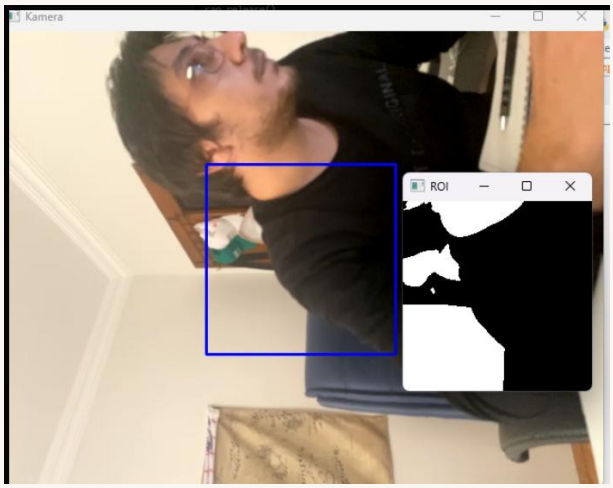
The first version ran on the desktop with OpenCV. Each webcam frame is shown in a Kamera window with a region-of-interest box; the area inside the box is converted to a clean black-and-white image — exactly what the model receives — and shown in a second ROI window. My laptop had no usable webcam, so I streamed my phone in as an external camera (DroidCam) to test it.

camera.py

```
import cv2

cap = cv2.VideoCapture(2) # external camera (phone via DroidCam)
if not cap.isOpened():
    print('Camera could not be opened!')
    exit()

while True:
    ret, frame = cap.read() # grab a frame
    if not ret:
        break
    cv2.imshow('Camera', frame) # show the live feed
    if cv2.waitKey(1) & 0xFF == ord('q'): # press q to quit
        break

cap.release()
cv2.destroyAllWindows()
```



Live webcam feed with the region-of-interest box (OpenCV “Kamera” window).



The ROI window shows the binarised digit the model actually sees.

ERRORS & DEBUGGING

The bugs I had to get past

Getting the deep-learning stack to run was the hardest part of the whole project. Across several machines and Python versions, the very first line of the program kept failing — TensorFlow simply would not import:

ERROR · terminal

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    import tensorflow as tf
ModuleNotFoundError: No module named 'tensorflow'
```

The root cause was version mismatch: TensorFlow only ships wheels for certain Python versions. The fix was to pin a clean environment — Python 3.10 in a dedicated virtual environment — and reinstall the stack with pip install tensorflow opencv-python numpy.

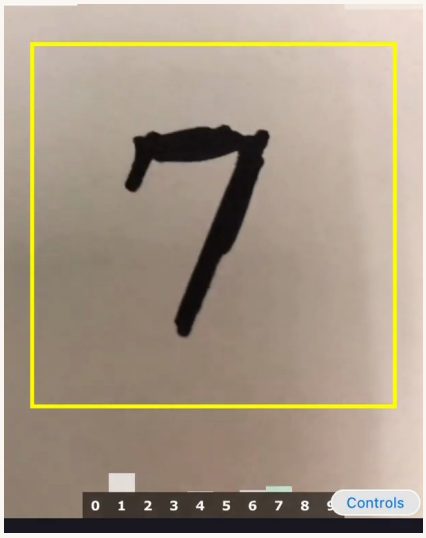
CHALLENGES & HOW I SOLVED THEM

- Python ↔ TensorFlow versions. The import above failed on several Python versions; after a lot of trial and error I standardised on Python 3.10 inside a virtual environment, which fixed it for good.
- Camera input. With no built-in webcam, I routed my phone's camera in over DroidCam and tuned the ROI cropping and thresholding so digits read cleanly under normal lighting.
- Stability & sharing. The desktop inference loop was occasionally flaky and hard for others to run, so I re-built the recogniser as a browser web app — which became the stable, install-free version anyone can open from a link.

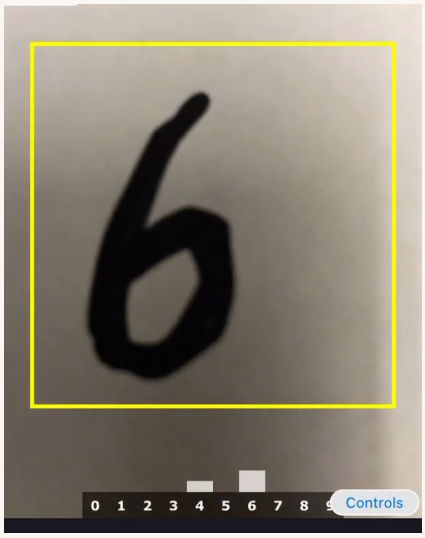
THE WEB APP

Reading real digits, in the browser

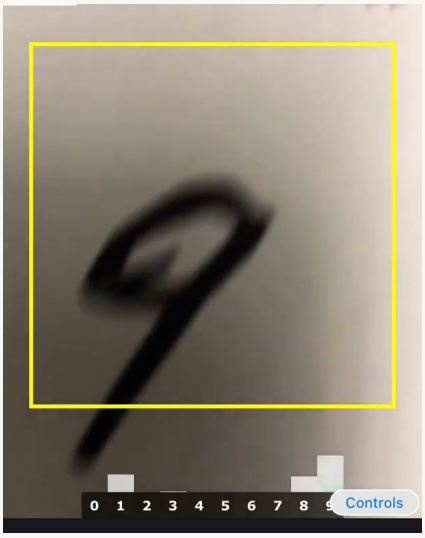
The deployed web app runs entirely on a phone or laptop — no install needed. Hold a handwritten digit inside the yellow frame and the network responds instantly, with a bar chart showing how confident it is about each possible digit.



Recognising a 7



Recognising a 6



Recognising a 9

RESULTS

What it achieves

- ~98% accuracy on the MNIST test set with a lightweight CNN that trains in minutes.
- Real-time inference — predictions update live as the digit moves in frame.
- Full confidence read-out — a 0–9 probability bar makes the model's reasoning visible, not just a single guess.
- Deployed & shareable — runs as a public web app on Netlify, no install required.

WHAT I LEARNED

Building the model was the easy part; the real lessons came from the pipeline around it — matching Python and TensorFlow versions, taming camera input and preprocessing, and turning a trained .h5 model into something a stranger can open in a browser and use.

BUILT WITH

Python	TensorFlow / Keras	OpenCV	NumPy	Matplotlib	CNN	MNIST	Netlify