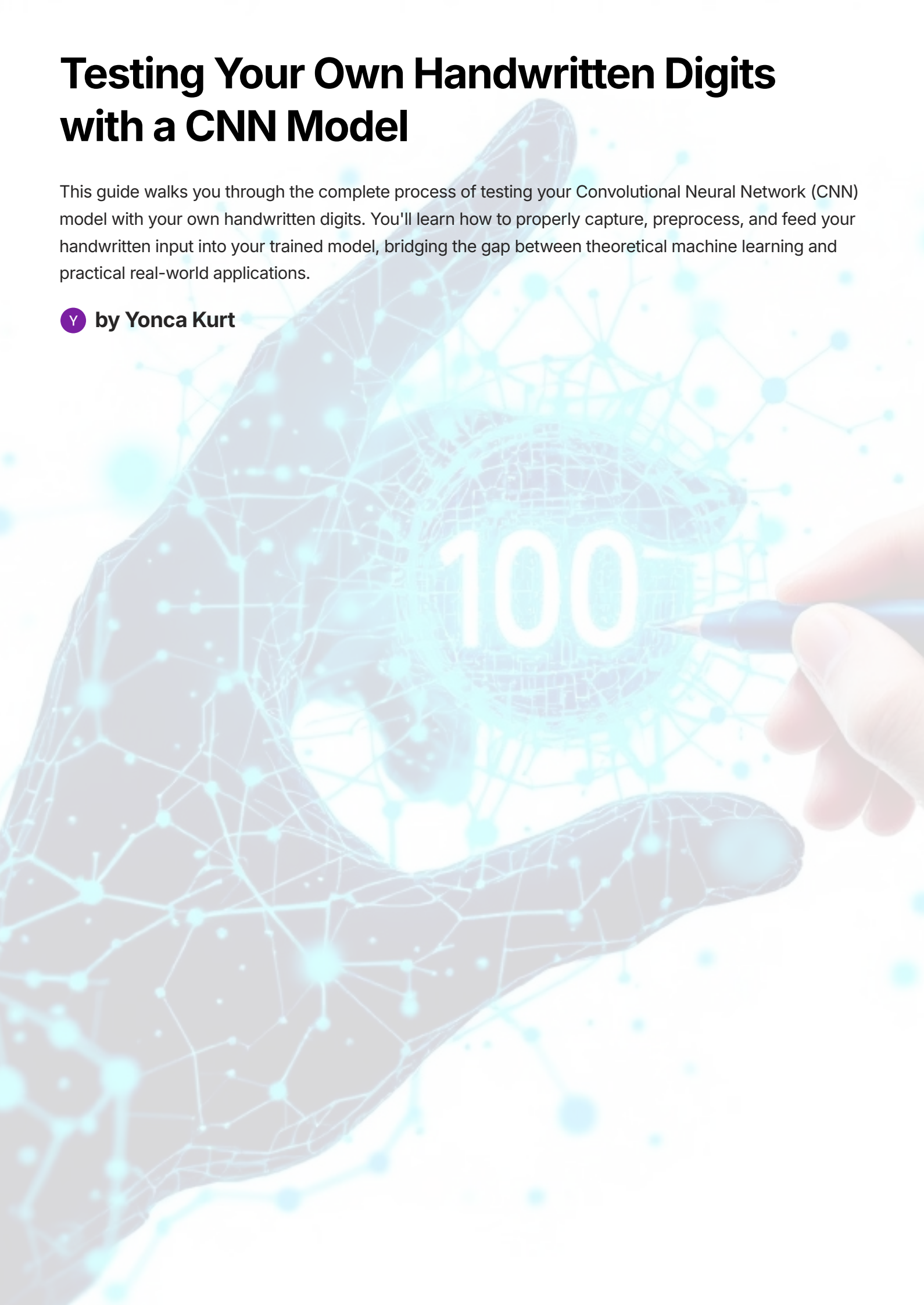
Testing Your Own Handwritten Digits with a CNN Model

This guide walks you through the complete process of testing your Convolutional Neural Network (CNN) model with your own handwritten digits. You'll learn how to properly capture, preprocess, and feed your handwritten input into your trained model, bridging the gap between theoretical machine learning and practical real-world applications.

Y **by Yonca Kurt**



Project Overview and Objectives

This project allows you to test the real-world applicability of your CNN model by using it to recognize digits that you've physically written yourself. Unlike working with the pre-processed MNIST dataset, this exercise introduces you to the challenges of preparing real-world inputs for machine learning models.

Learn preprocessing requirements

Understand how raw images must be transformed to match model expectations

Test model robustness

Evaluate how well your CNN performs on inputs outside the training dataset

Apply practical ML skills

Experience the full pipeline from image capture to prediction

By completing this exercise, you'll gain valuable insight into how preprocessing affects model performance and how to bridge the gap between theoretical models and practical applications.

Step 1: Drawing and Capturing Your Digit

Materials Needed

- White A4 paper
- Dark pencil or marker
- Smartphone camera or webcam

Key Tips

- Draw the digit clearly and boldly
- Ensure good lighting when taking the photo
- Center the digit in the frame
- Avoid shadows or glare on the paper

When capturing your digit, aim for a clean, high-contrast image. The digit should be prominent in the frame and captured under good lighting conditions to minimize preprocessing challenges later.

After capturing, save the image in either PNG or JPG format with a descriptive filename that includes the digit you've drawn for easy reference.

Step 2: Preprocessing Your Image

Before your handwritten digit can be fed into the CNN model, it must be preprocessed to match the format of the MNIST dataset used for training. This crucial step transforms your raw image into the exact specifications expected by the neural network.

1

Load and Convert to Grayscale

Use the PIL library to open your image and convert it to grayscale mode, which reduces the color information to a single channel.

```
from PIL import Image, ImageOps
import numpy as np

img = Image.open('your_image.png').convert('L')
```

2

Invert Colors (If Necessary)

If your digit is white on a black background (opposite of MNIST standard), you'll need to invert the colors.

```
# Only use if digit is white on black background
# img = ImageOps.invert(img)
```

3

Resize and Normalize

Resize the image to 28x28 pixels and normalize pixel values to the range [0,1].

```
img = img.resize((28, 28))
img_array = np.array(img) / 255.0
```

4

Reshape for Model Input

Reshape the array to match the input dimensions expected by the CNN model.

```
img_array = img_array.reshape(1, 28, 28, 1)
```

Step 3: Making a Prediction

Once your image has been properly preprocessed, you can feed it into your CNN model to make a prediction. This is where you'll see if your model can successfully recognize your handwritten digit.

```
# Predict using the trained CNN model
prediction = model.predict(img_array)

# Get the predicted digit
predicted_label = np.argmax(prediction)

print("Predicted Label:", predicted_label)
```

The **predict()** method returns an array of probabilities for each possible digit (0-9).

The **np.argmax()** function then identifies which digit has the highest probability, giving you the final prediction.

Your model will analyze the preprocessed image and determine which digit it most closely resembles based on the patterns it learned during training. The confidence of the prediction can vary depending on how similar your handwriting is to the MNIST training examples.

Troubleshooting Common Issues

Incorrect Preprocessing

- Forgetting to invert colors when necessary
- Incorrect image dimensions
- Not normalizing pixel values to [0,1]

Solution: Double-check each preprocessing step, and visualize the image after each transformation to ensure it looks similar to MNIST digits.

Poor Image Quality

- Low contrast between digit and background
- Shadows or glare in the original photo
- Multiple digits or extraneous marks in the image

Solution: Recapture the image under better lighting with a bolder digit, or apply additional preprocessing to enhance contrast.

Model Performance Issues

- Low accuracy on handwritten digits
- Consistently misclassifying certain digits

Solution: Consider retraining your model with data augmentation to improve robustness, or adjust your handwriting style to more closely match MNIST patterns.

Remember that even well-trained models may struggle with handwriting styles significantly different from those in the MNIST dataset. Experimenting with different writing styles can help you understand your model's strengths and limitations.

Bonus: Visualizing Your Results

To better understand how your model is interpreting your input, you can create a visualization that displays both your preprocessed image and the model's prediction.

```
import matplotlib.pyplot as plt

# Reshape for display (remove batch and channel dimensions)
display_image = img_array.reshape(28, 28)

# Create the visualization
plt.figure(figsize=(5, 5))
plt.imshow(display_image, cmap='gray')
plt.title(f"Predicted: {predicted_label}")
plt.axis('off')
plt.show()
```

This visualization helps you see exactly what your model is "seeing" after preprocessing. If your prediction is incorrect, this can provide valuable clues about what might be going wrong. For example, you might notice that parts of your digit got cut off during resizing, or that the contrast isn't as clear as it should be.

For more advanced visualization, you could also display the confidence levels for each possible digit by adding a bar chart showing the probabilities returned by the model.predict() function.

Learning Outcomes

Understanding Preprocessing

Applied techniques to transform raw inputs for AI models

Application Readiness

Prepared for practical CNN implementation

Full ML Pipeline

Implemented end-to-end workflow from capture to prediction

Real-world Testing

Tested model on inputs outside training data

Next Steps: What's Possible Now?

1

Expand Inputs

Test different writing tools and backgrounds

2

Mobile App

Develop with camera integration

3

Custom Dataset

Create your own from scratch

This project bridges theory and practice in deep learning. You've learned to adapt models to real-world inputs and understand preprocessing's vital role. These skills transfer directly to more complex vision projects and AI applications.