Intelligent tactile imaging-recognition sensor system enabled by methoxynitrobenzene-salicylaldehyde fluorescent material

Zihan Liu, Xinyi Zhao, Yuai Duan, Yaping Li, Zhijia Wang, Zixuan Wang, Jiarong Zhang, Jing Yuan, Hua Geng, Tianyu Han*

Department of Chemistry, Capital Normal University, Beijing, China, 100048.

*Corresponding author. E-mail address: hanty@cnu.edu.cn.

Table of contents

1. Materials and instruments

The involved chemicals and organic solvents were provided by Macklin and J&K Scientific, respectively. They were used as received, without purification. ^{1}H and ^{13}C NMR spectra of **MNIMP** were obtained by a 600 MHz Varian-VNMRS nuclear magnetic resonance spectrometer with tetramethylsilane (TMS) internal reference. Melting point of **MNIMP** was measured by a Hanon-MP420 (digital version) melting point apparatus. The high-resolution mass spectrum (HRMS) of **MNIMP** was measured by Xevo^{TM} G2-XS TOF mass spectrometer in positive ionization mode. UV-vis spectra of **MNIMP** were recorded by a SHIMADZU UV-2550 spectrophotometer. Emission spectra of **MNIMP** were recorded by a HITACHI F-7000 spectrofluorophotometer. Fluorescence images of **MNIMP** film were captured by an Olympus-CKX41 fluorescence microscope using 365 nm irradiation. The average thickness of **MNIMP** film was measured by a Smart-Sensor (AS-931) coating thickness gauge. Fluorescent images for texture recognition were taken with a Xiaomi-13 smartphone. The robot arm used to build the automation platform is by Keyes Micro:bit. Thermogravimetric (TG) and derivative thermogravimetric (DTG) curves were measured by SHIMADZU DTG-60AH thermogravimetric meter. Powder X-ray diffraction patterns were recorded on Bruker D8 advance X-ray diffractometer (Bruker, Germany). Single crystal X-ray diffraction was performed on a Bruker D8 venture X-ray diffractometer (Bruker, Germany). The intensity values of the fluorescent patterns were measured by Image-Pro-Plus 6.0 software. The mean gray values of the fluorescent patterns were measured by Image-J 1.54f software.

2. Supplementary figures and tables

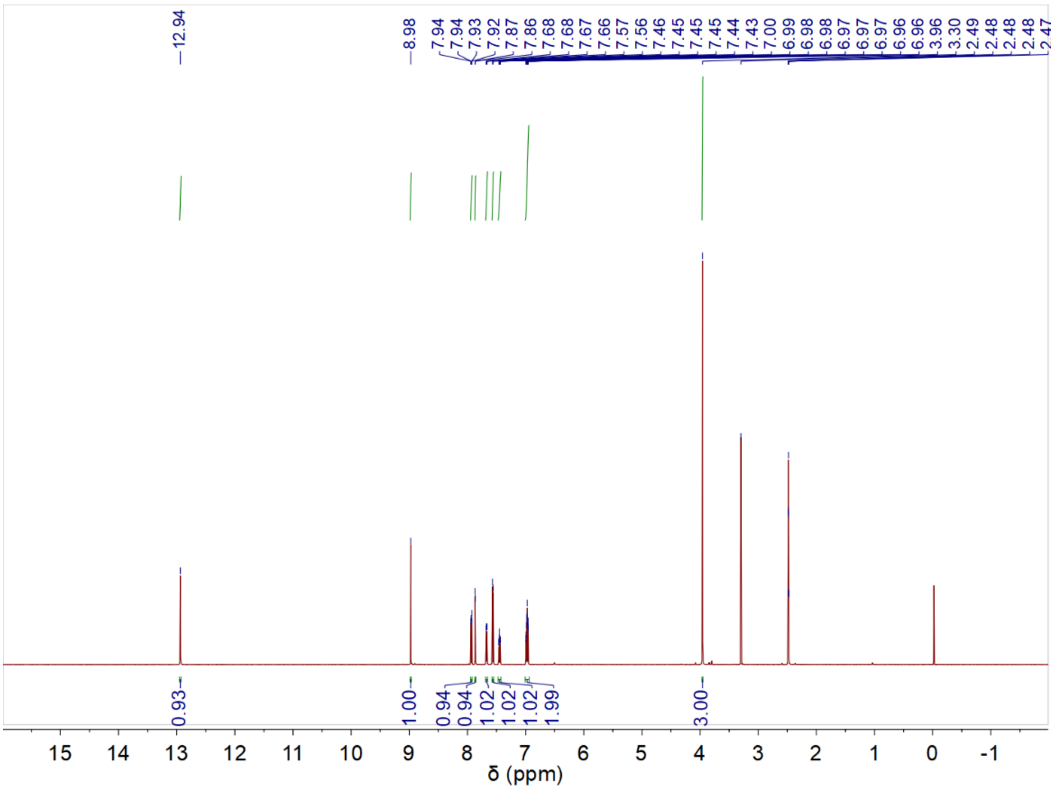


Figure S1. ^{1}H NMR (600 MHz, 298 K) spectrum of **MNIMP** in DMSO-d_6.

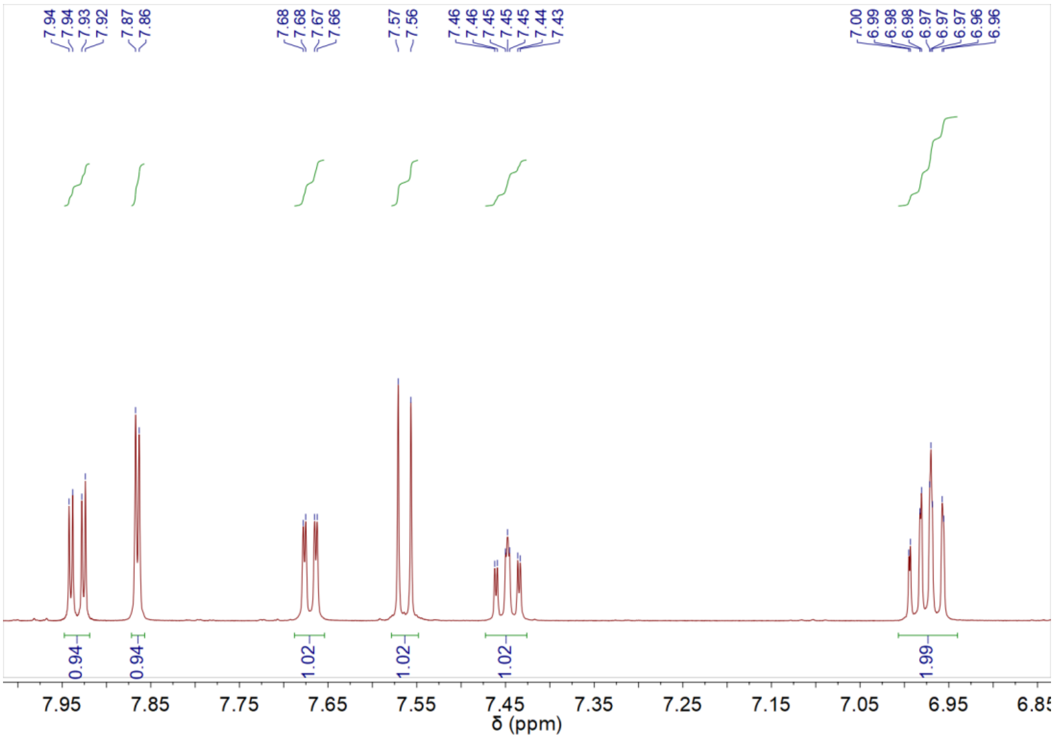


Figure S2. Magnified ^{1}H NMR spectrum of **MNIMP** (6.85-7.95 ppm) showing the peaks and splitting of the aromatic hydrogens.

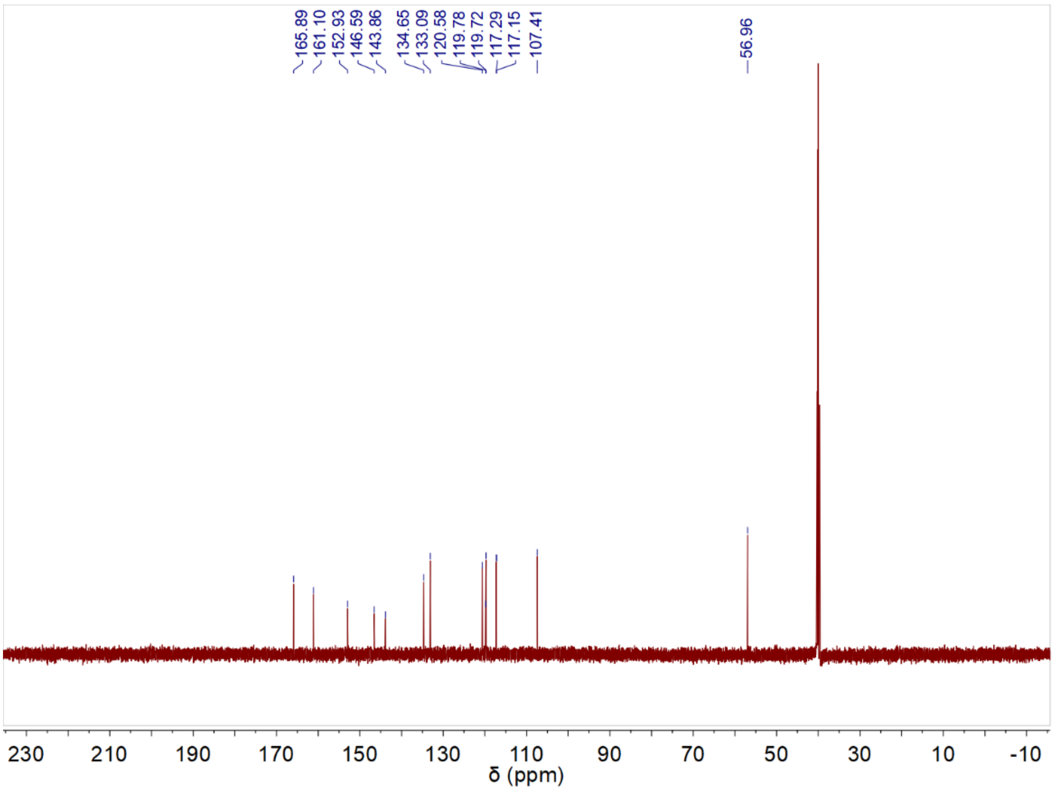


Figure S3. ^{13}C NMR (151 MHz, 298 K) spectrum of **MNIMP** in DMSO-d_6.

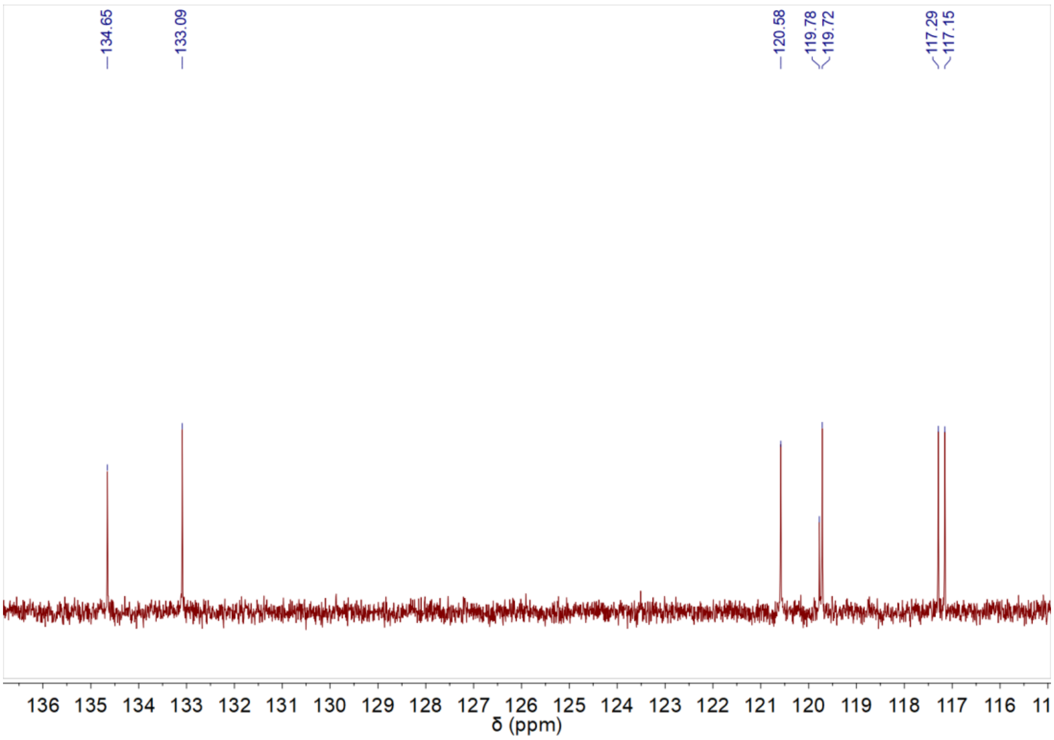


Figure S4. Magnified ^{13}C NMR spectrum of **MNIMP** ranging from 116-130 ppm.

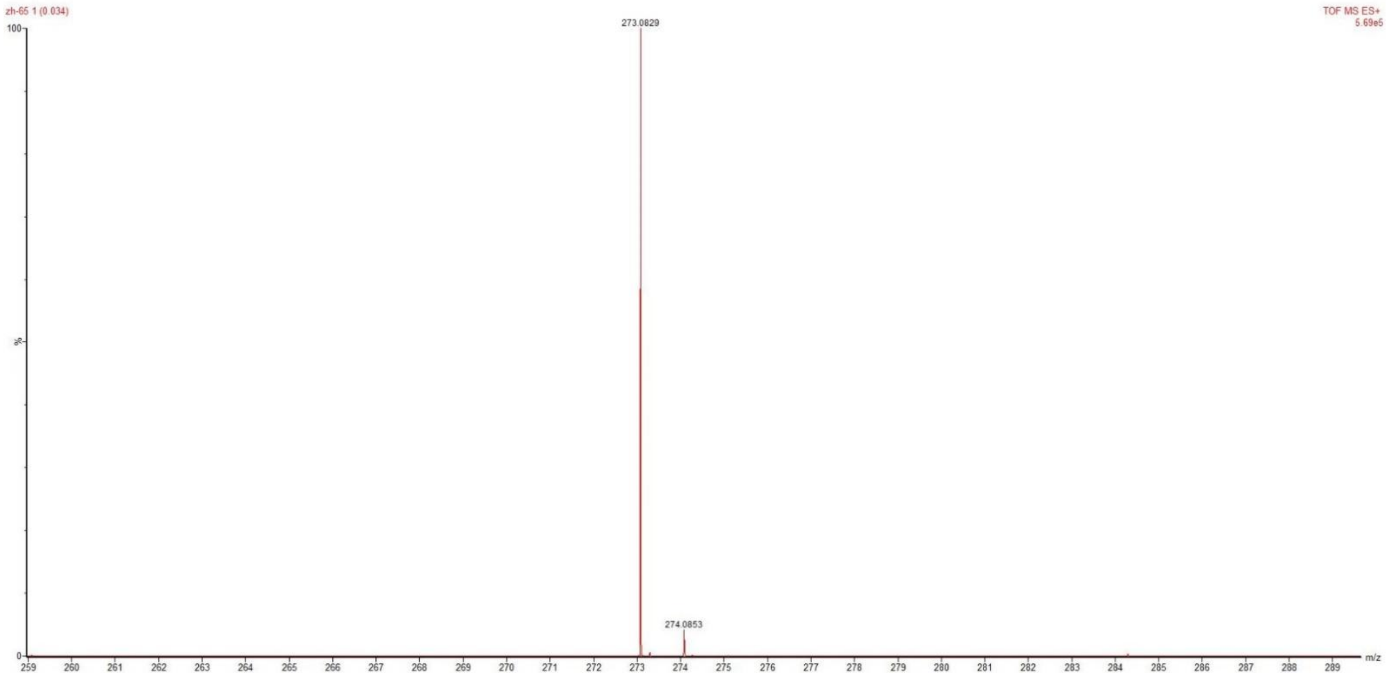


Figure S5. High resolution mass spectrum of **MNIMP**.

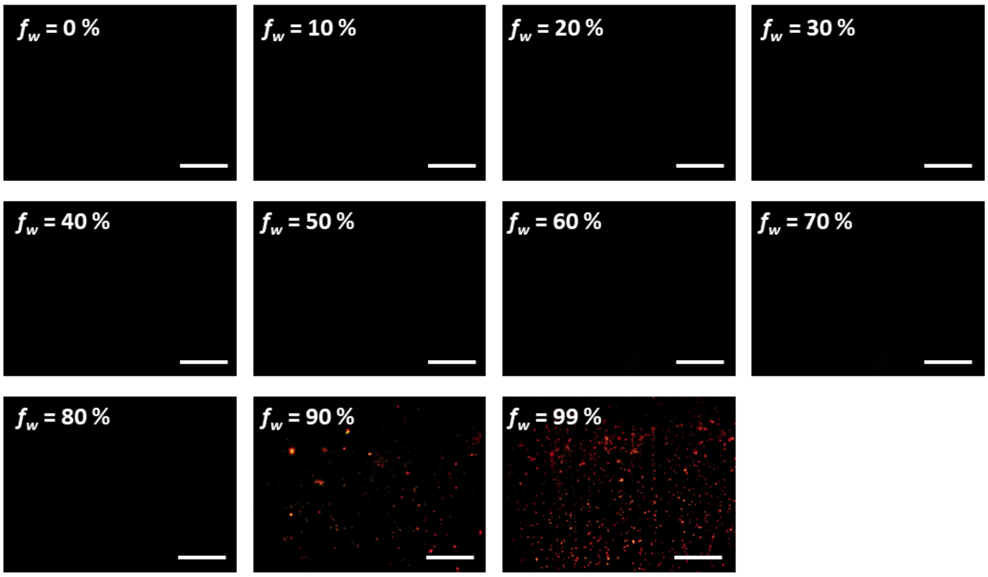


Figure S6. Fluorescence micrographs of **MNIMP** in hydrated states with varying water fraction (f_w: 0-99%). Scale bar: 200 µm

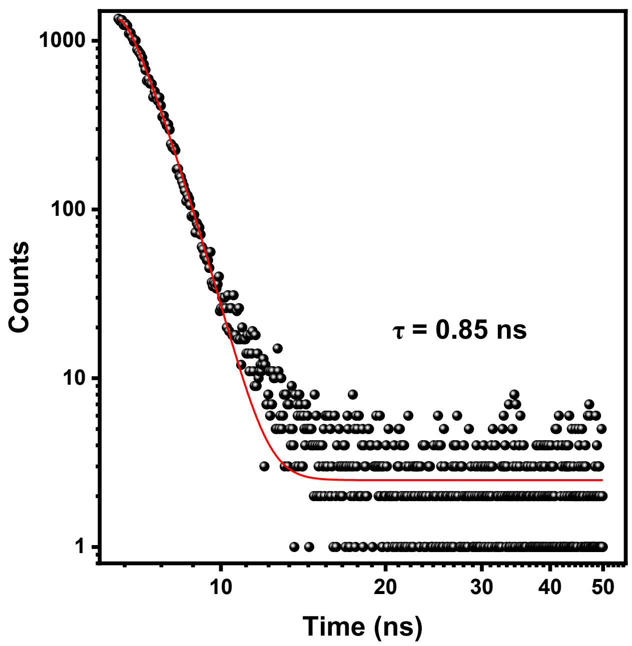


Figure S7. Fluorescence lifetime decay profile and the dual-exponential fitting curve of **MNIMP** film. Excitation wavelength: 375 nm.

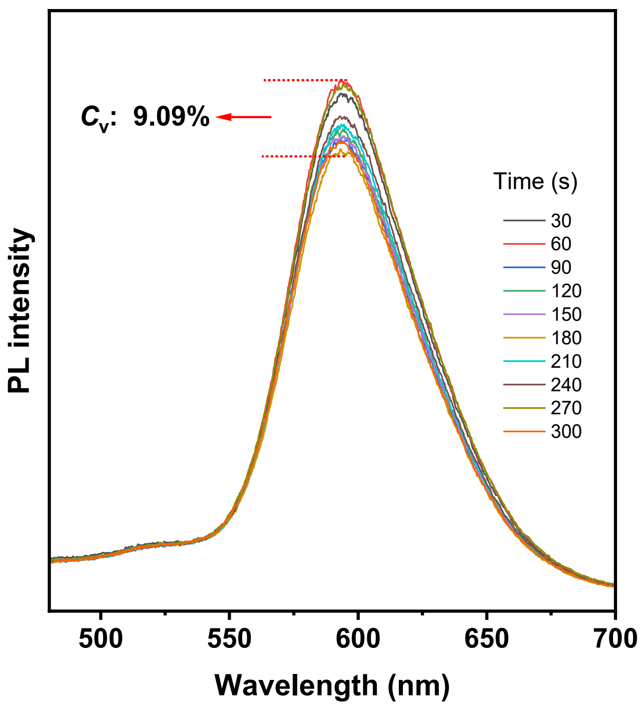


Figure S8. Time-dependent emission spectra of **MNIMP** with 1000 μW/cm^2 UV exposure (300 s). Excitation wavelength: 414 nm.

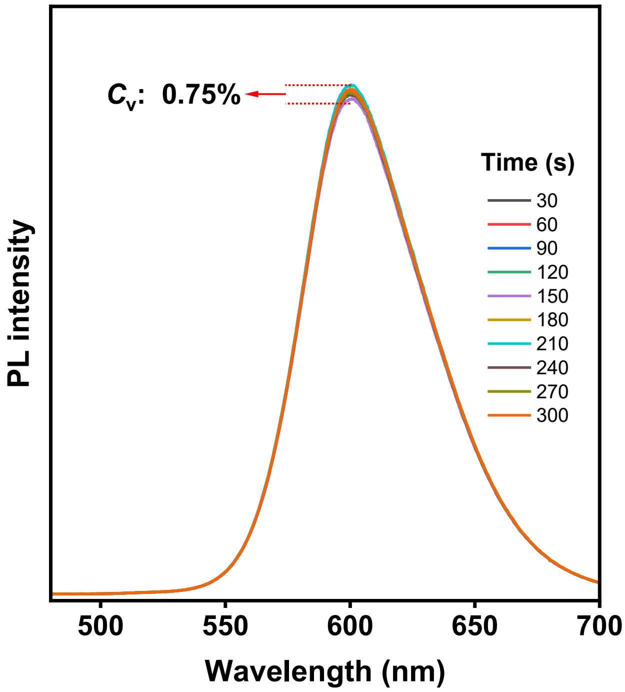


Figure S9. Time-dependent emission spectra of **MNIMP** with 90% RH humidity environment (300 s). Excitation wavelength: 414 nm.

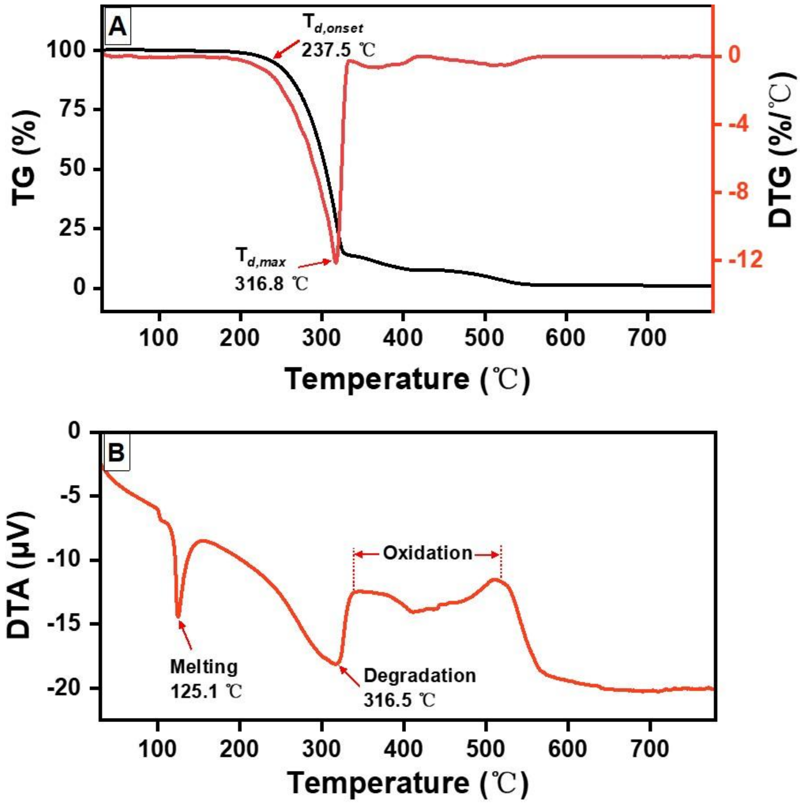


Figure S10. TG-DTG (A) and DTA (B) curves of **MNIMP** under air atmosphere at a heating rate of 10 °C/min.

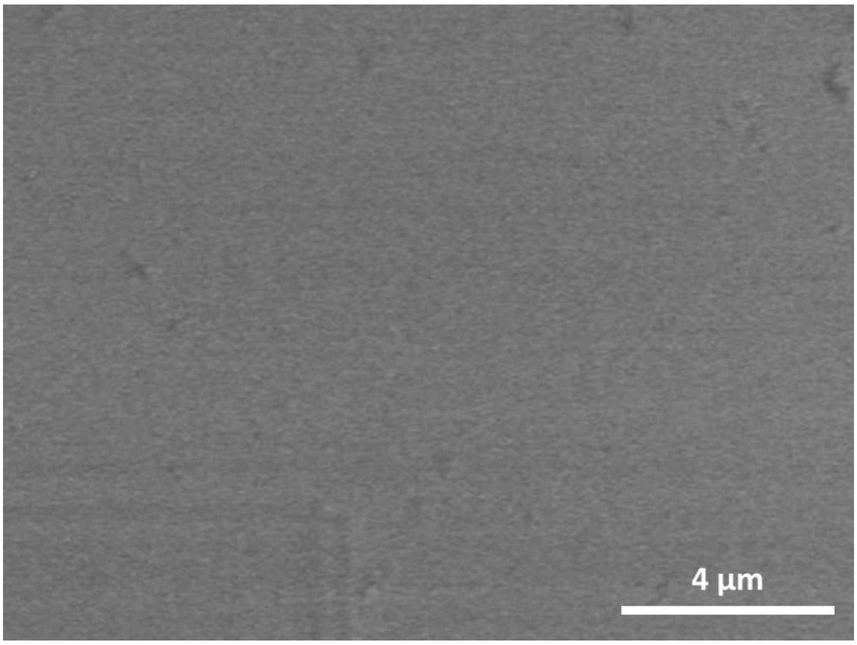


Figure S11. Scanning electron microscope (SEM) image of the as-preprepared film (before contact with target objects).

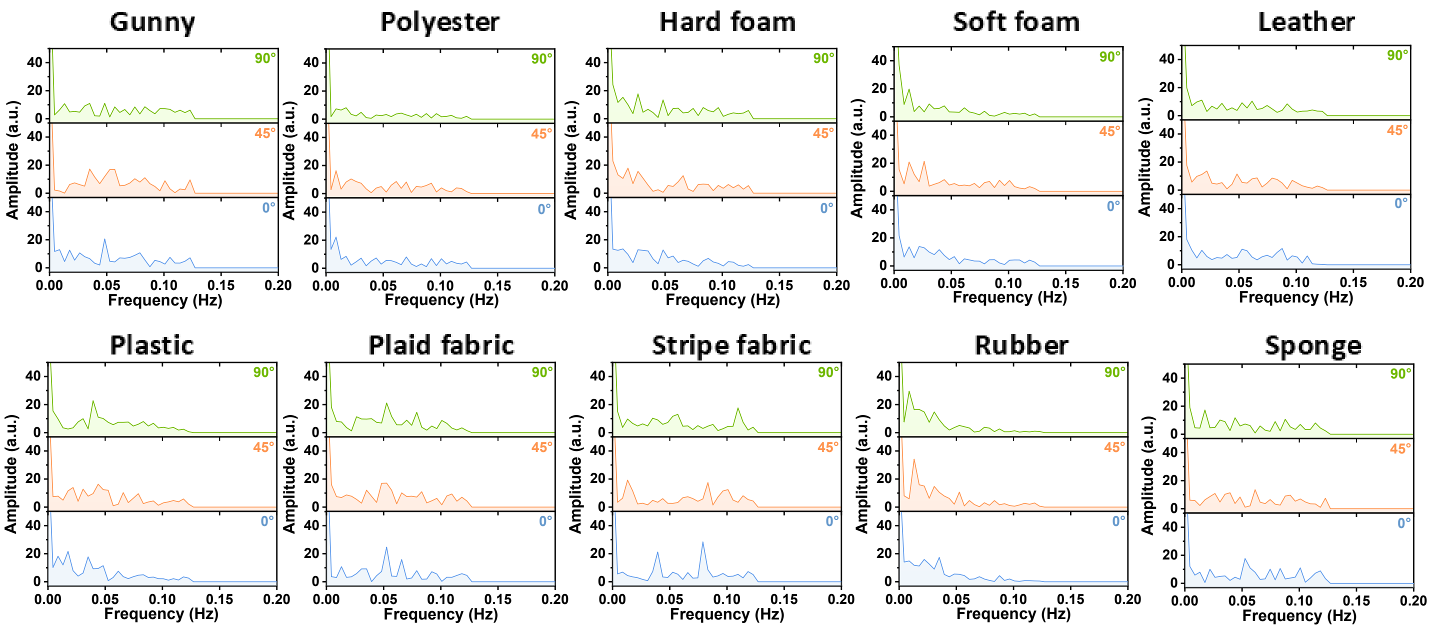


Figure S12. Fourier transform spectra of various textured targets in 0°, 45° and 90° directions (*i.e.*, gunny, polyester, hard foam, soft foam, leather, plastic, plaid fabric, stripe fabric, rubber and sponge).

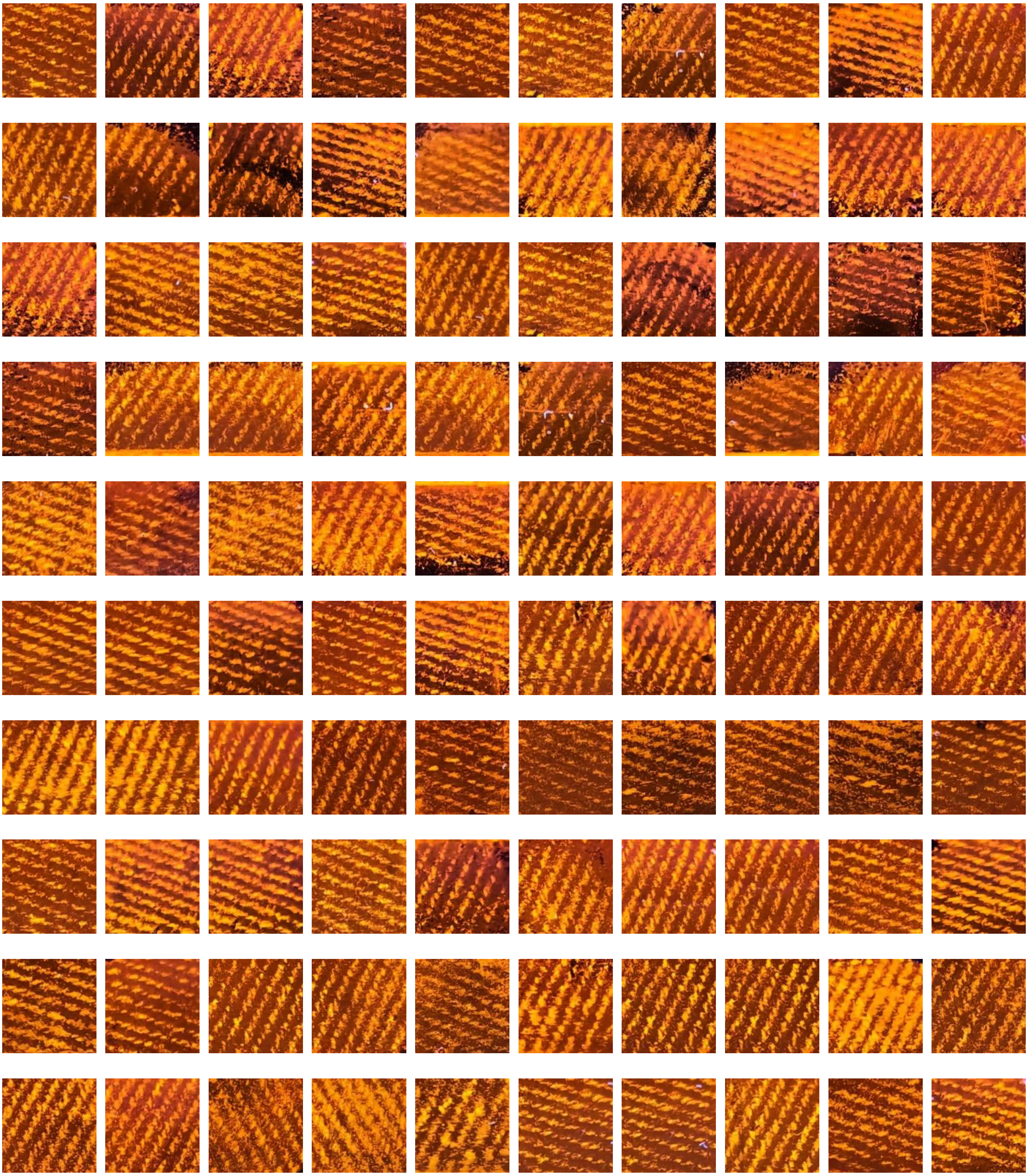


Figure S13. The image set of the surface textures of twill inputted into CNN model for training and recognition, containing 100 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation

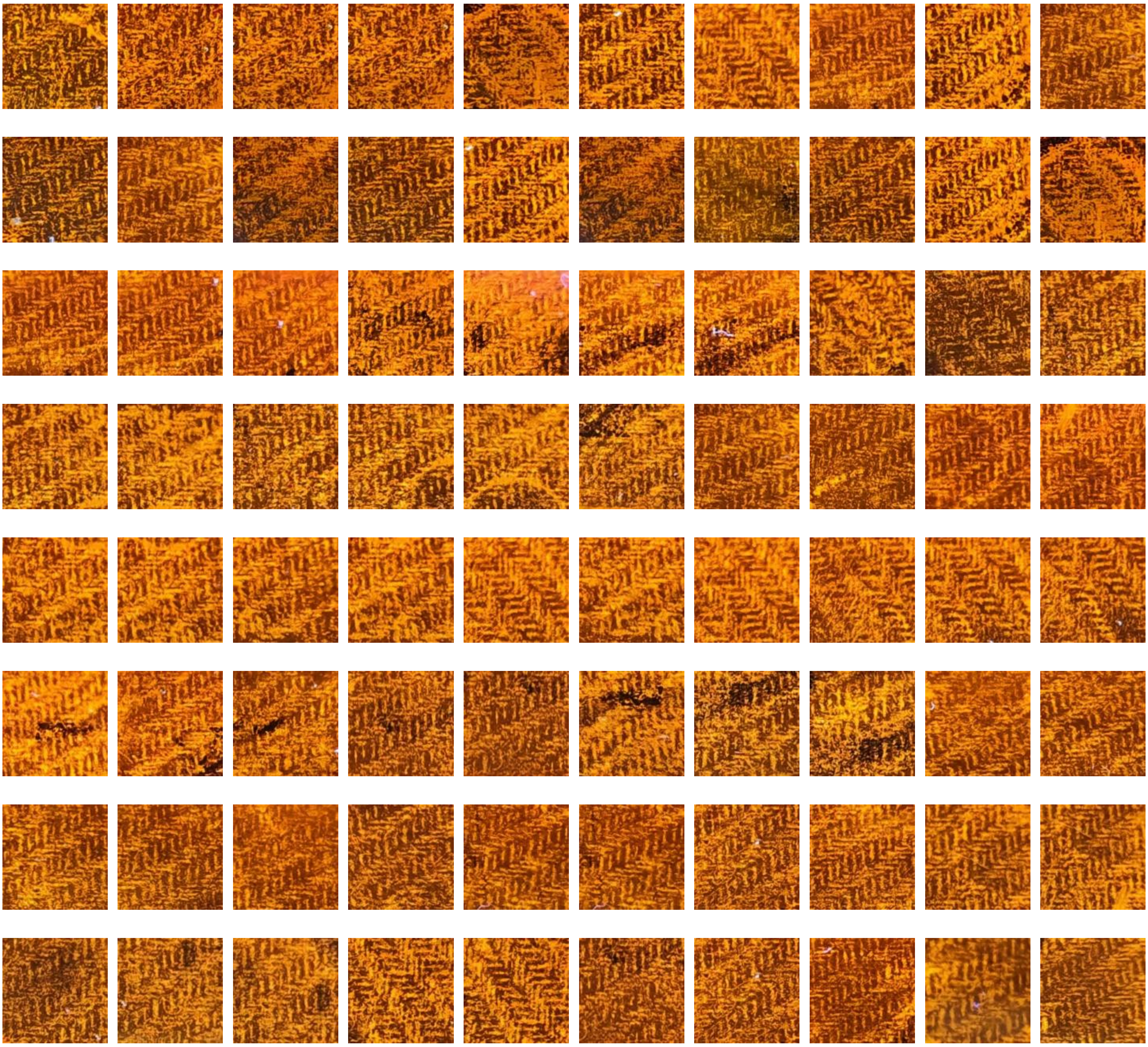


Figure S14. The image set of the surface textures of gunny inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

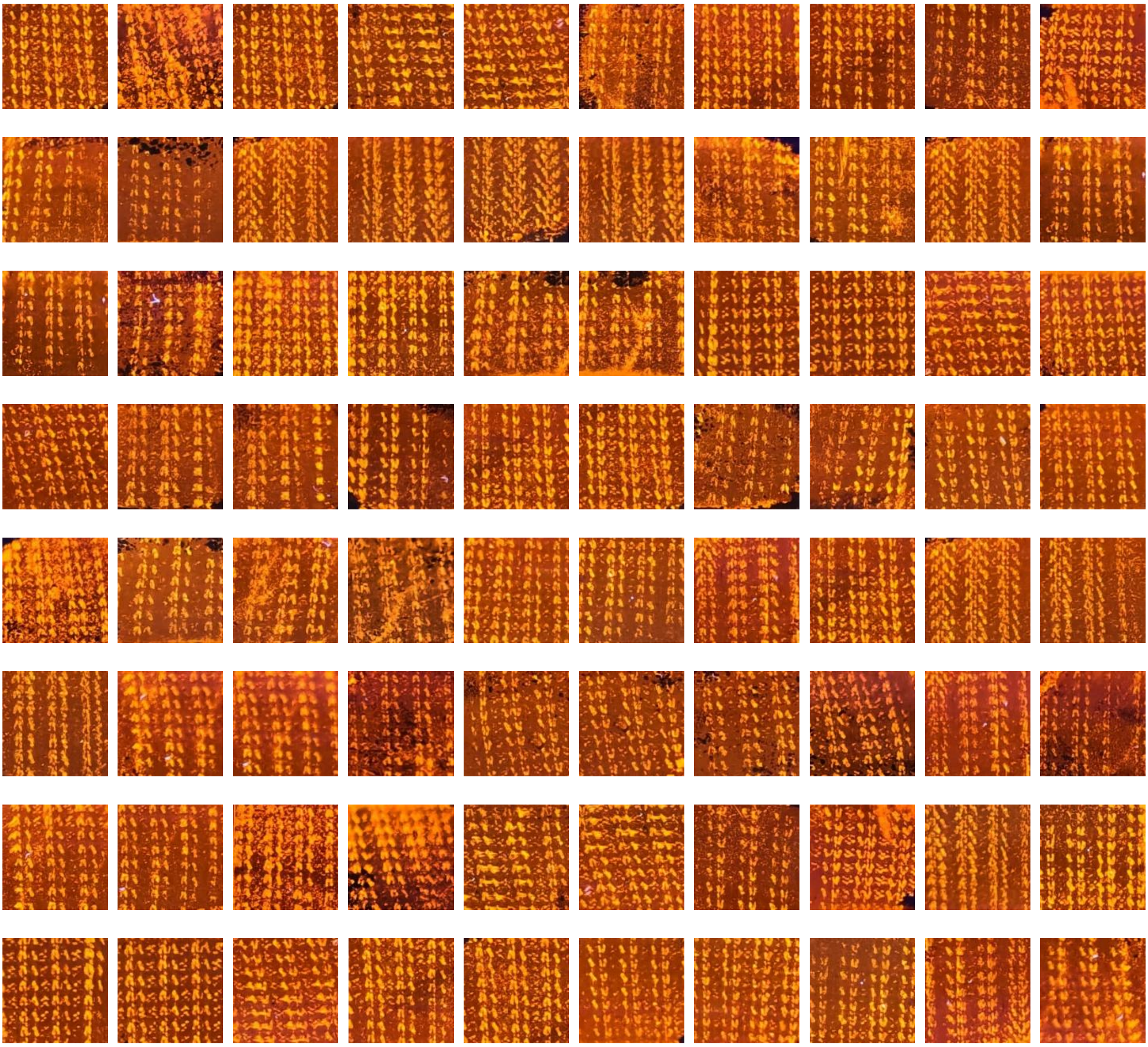


Figure S15. The image set of the surface textures of elastic inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

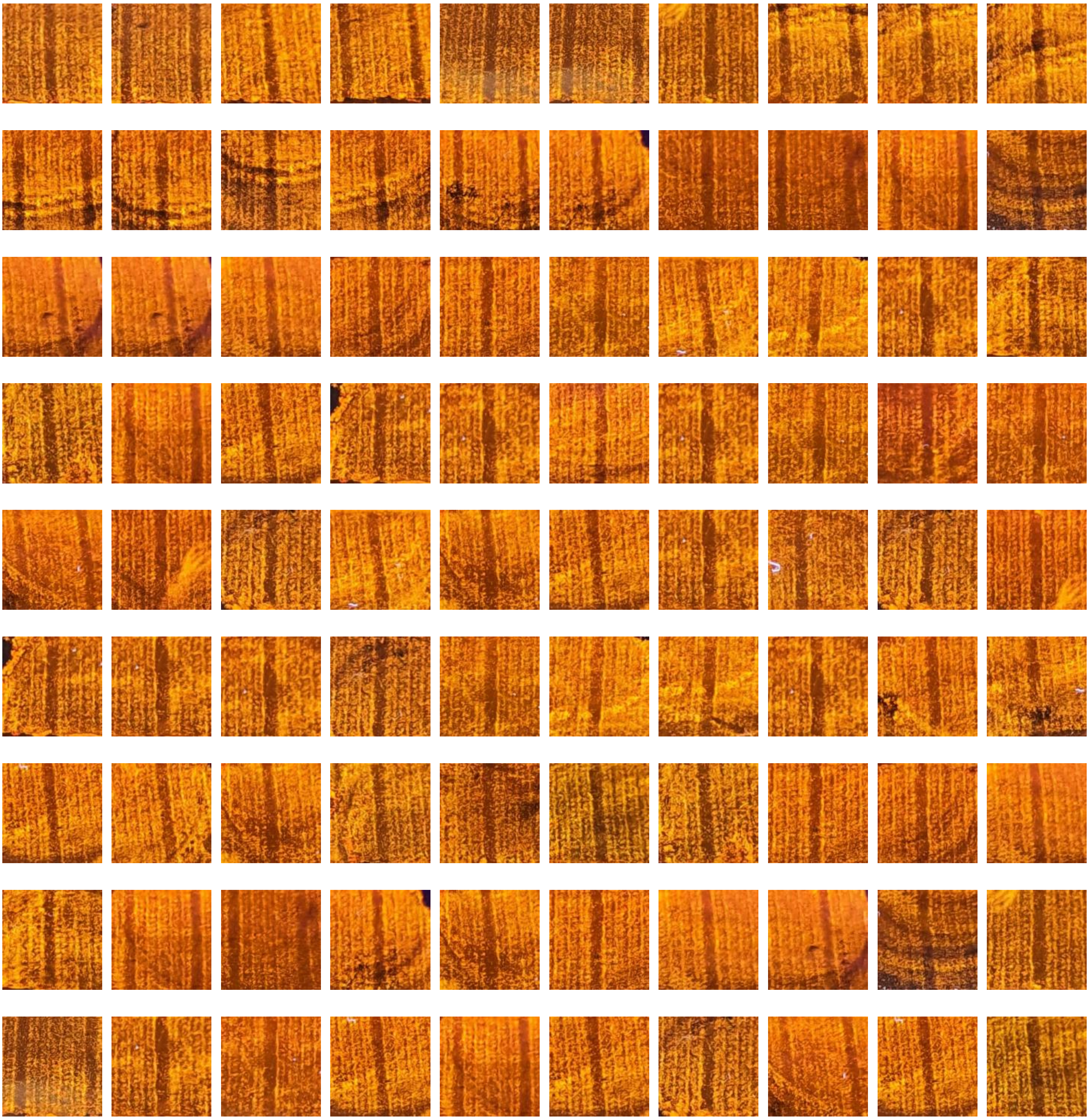


Figure S16. The image set of the surface textures of polyester inputted into CNN model for training and recognition, containing 90 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

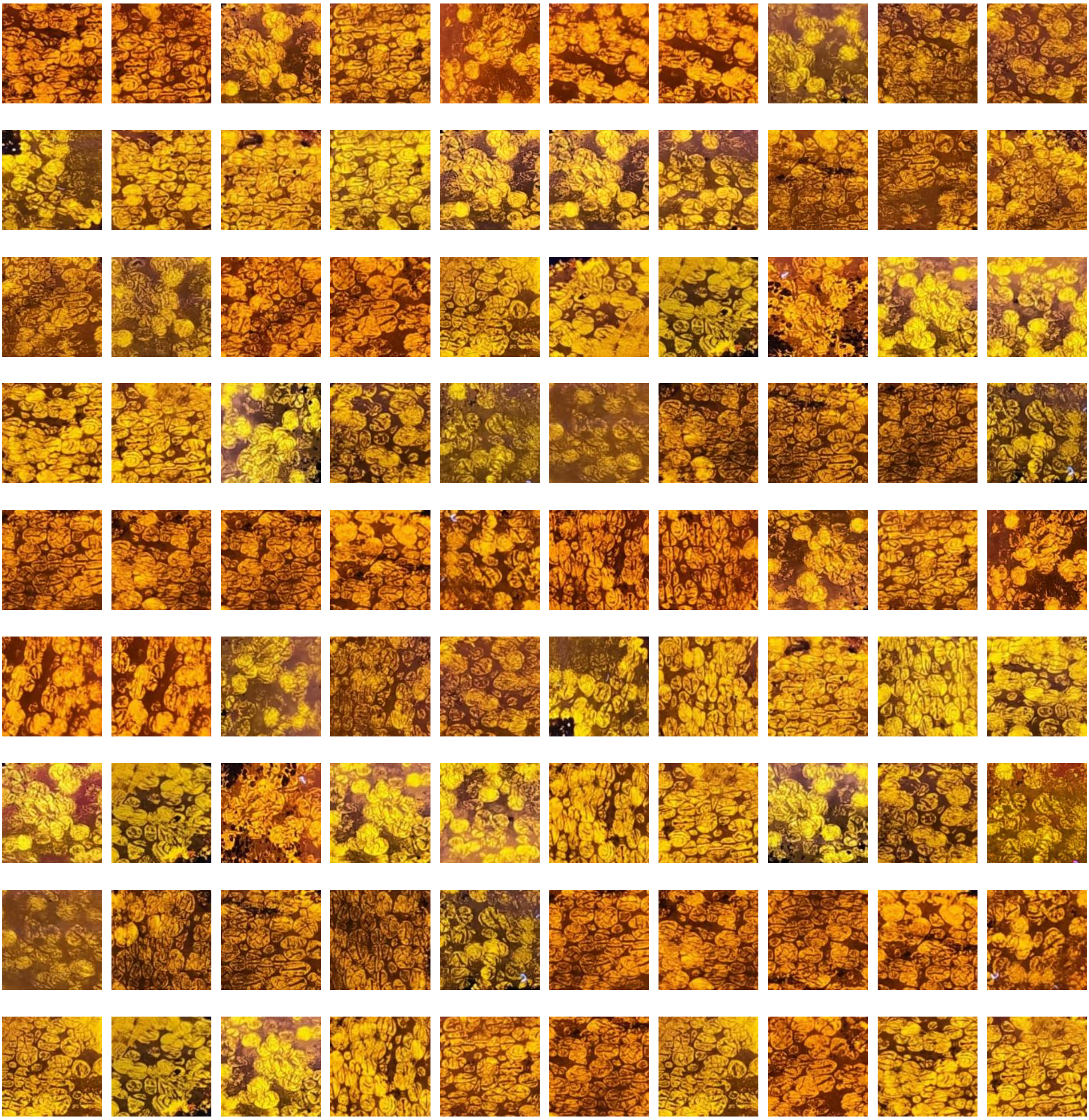


Figure S17. The image set of the surface textures of hard foam inputted into CNN model for training and recognition, containing 90 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

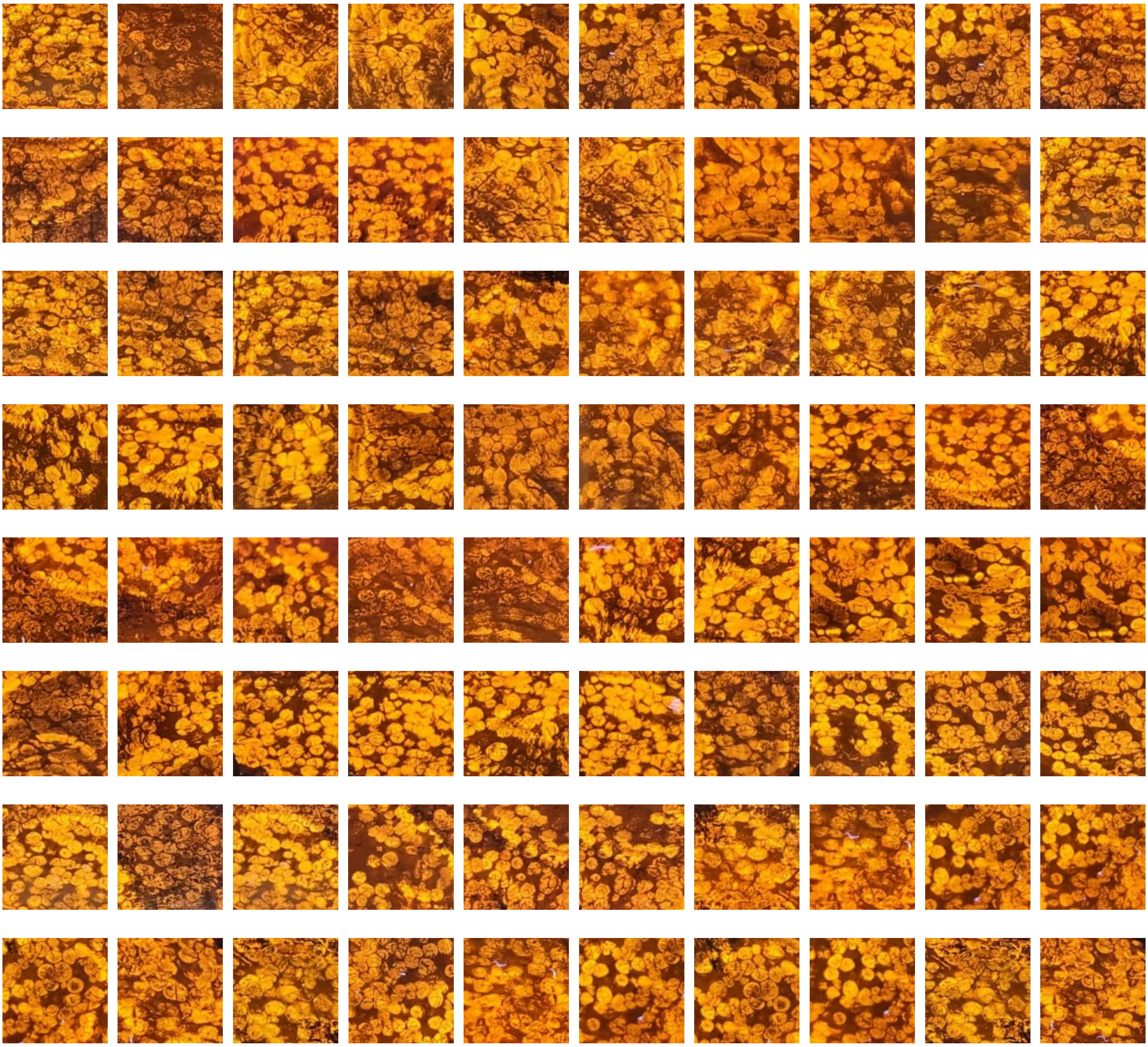


Figure S18. The image set of the surface textures of soft foam inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

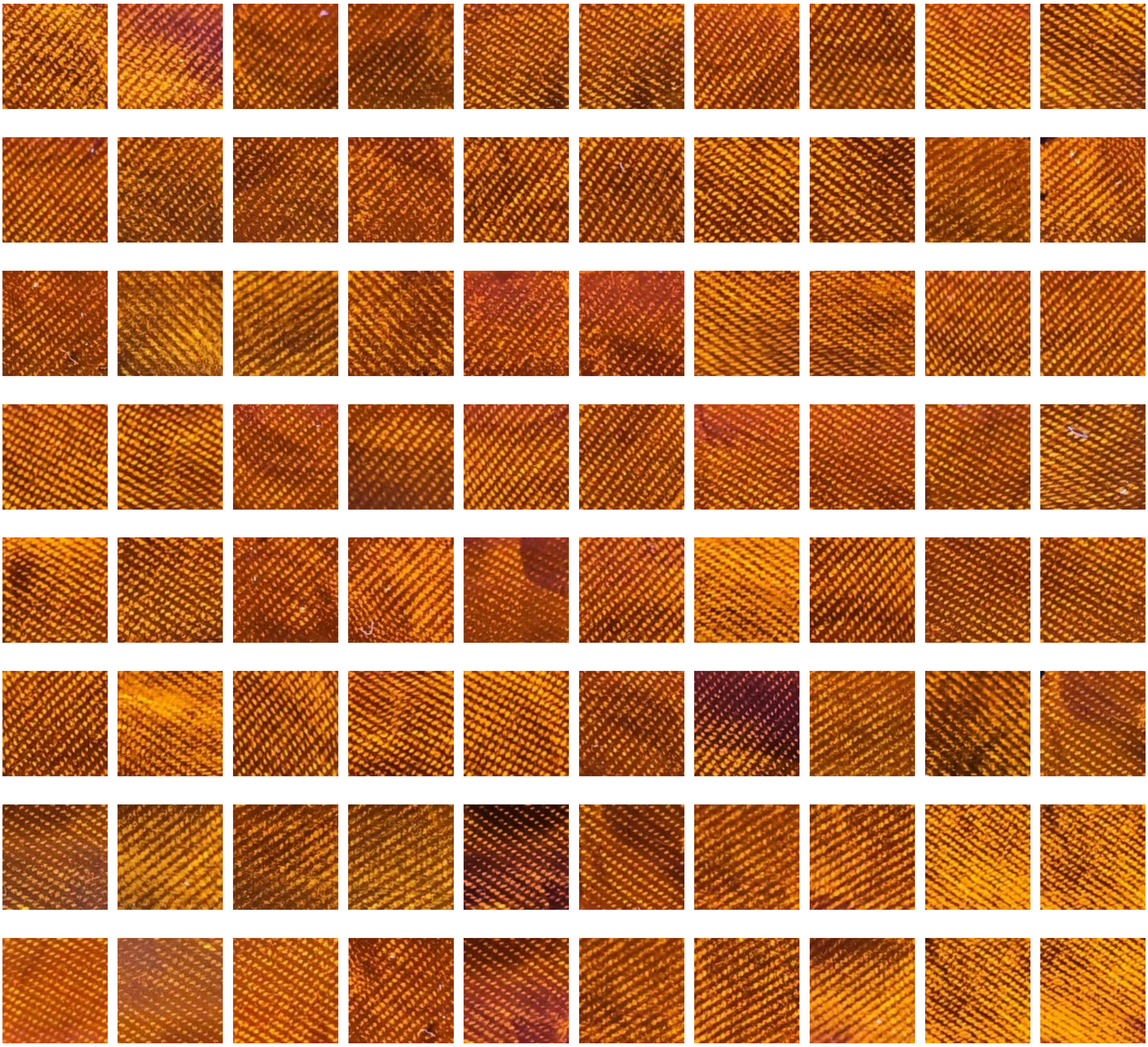


Figure S19. The image set of the surface textures of gauze inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

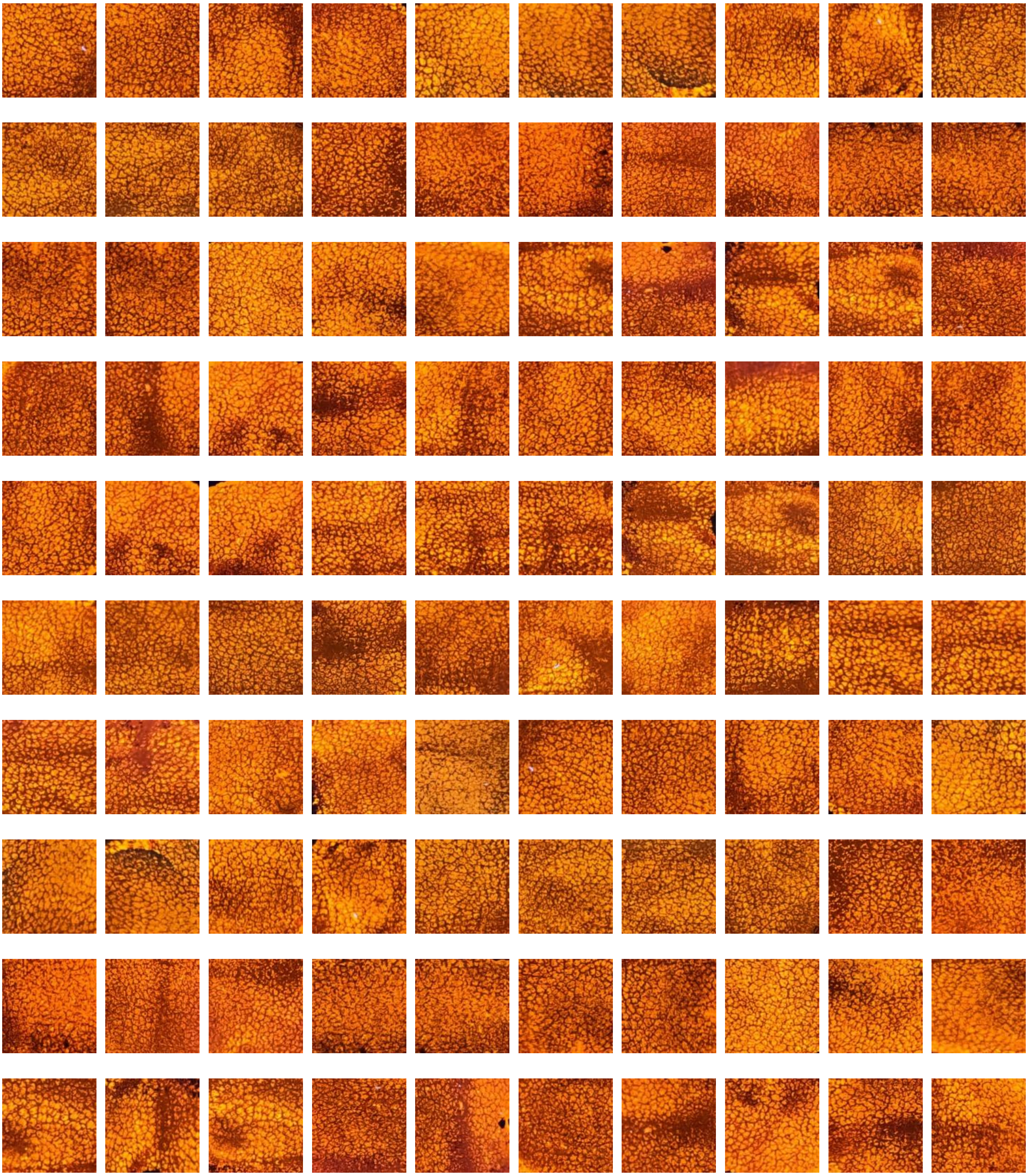


Figure S20. The image set of the surface textures of leather inputted into CNN model for training and recognition, containing 100 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

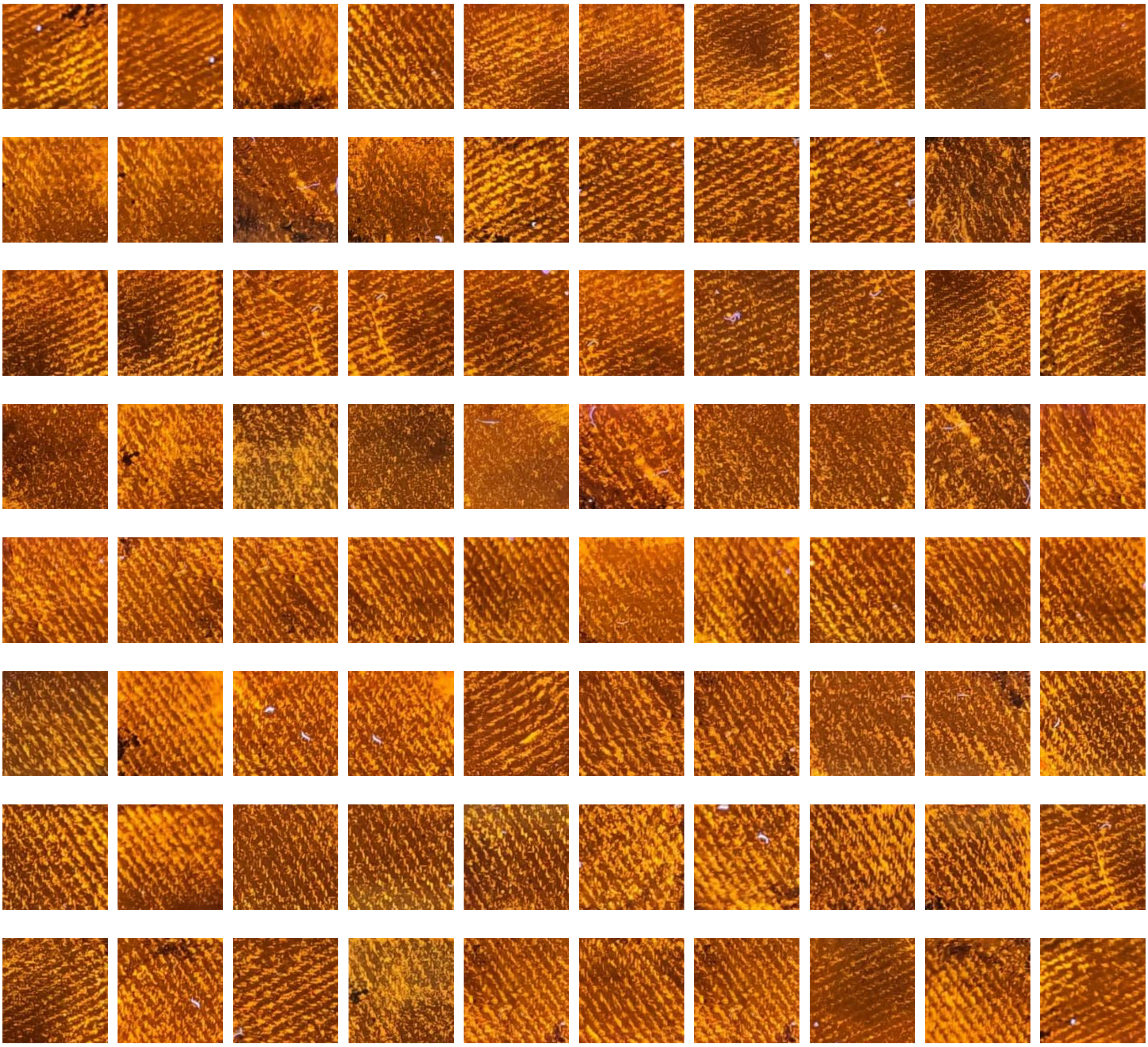


Figure S21. The image set of the surface textures of linen inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

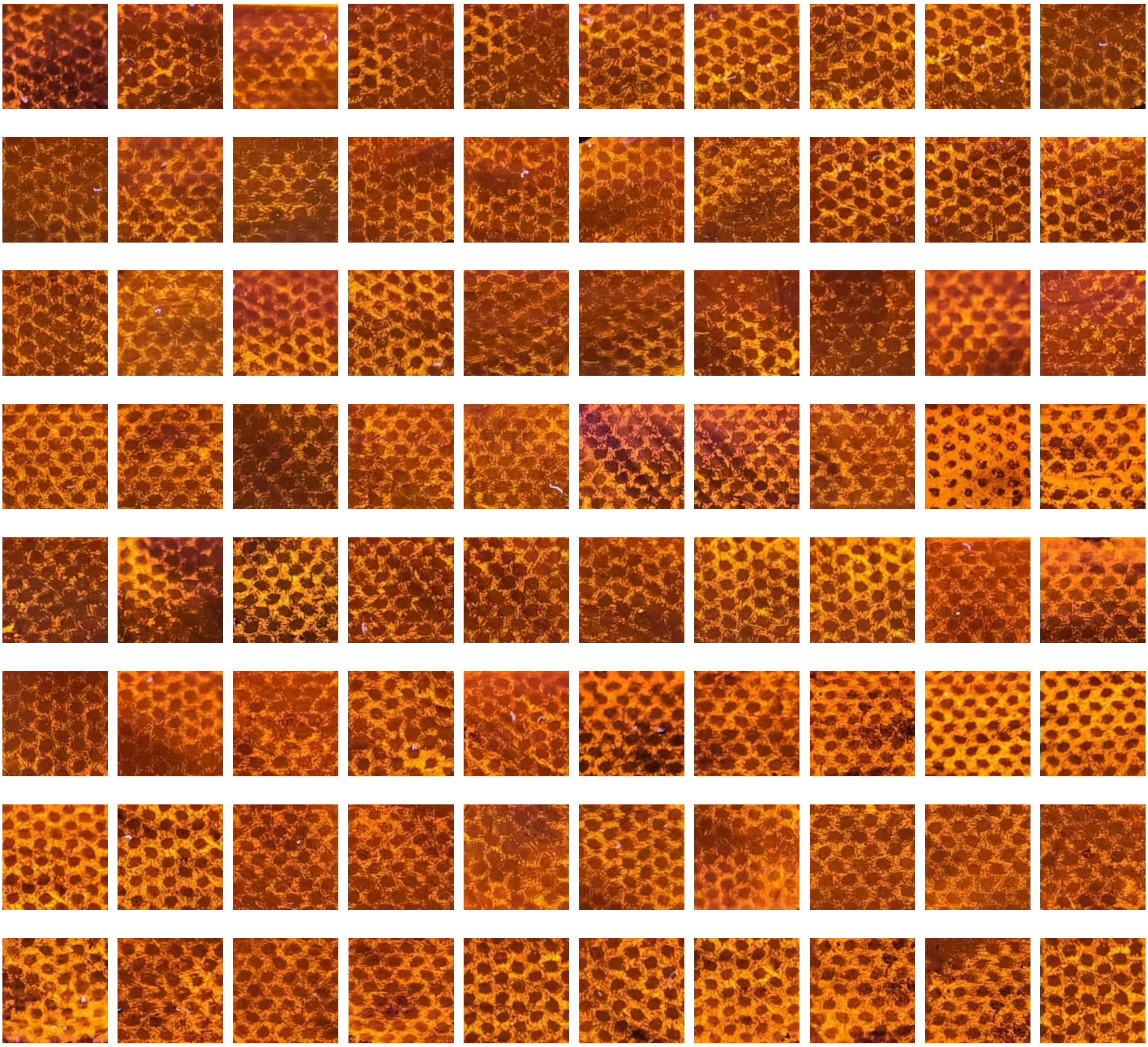


Figure S22. The image set of the surface textures of nonwoven inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

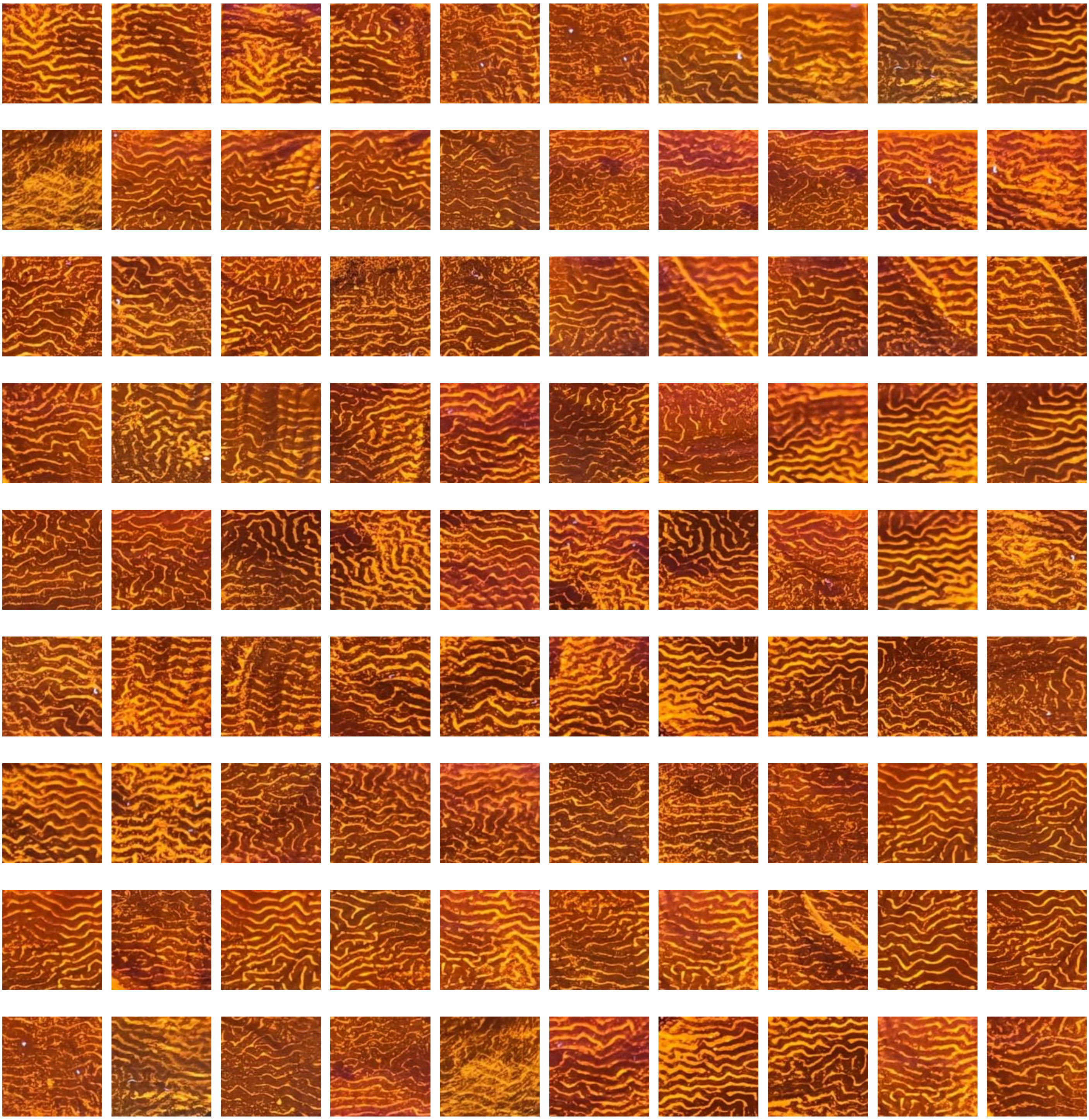


Figure S23. The image set of the surface textures of plastic inputted into CNN model for training and recognition, containing 90 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

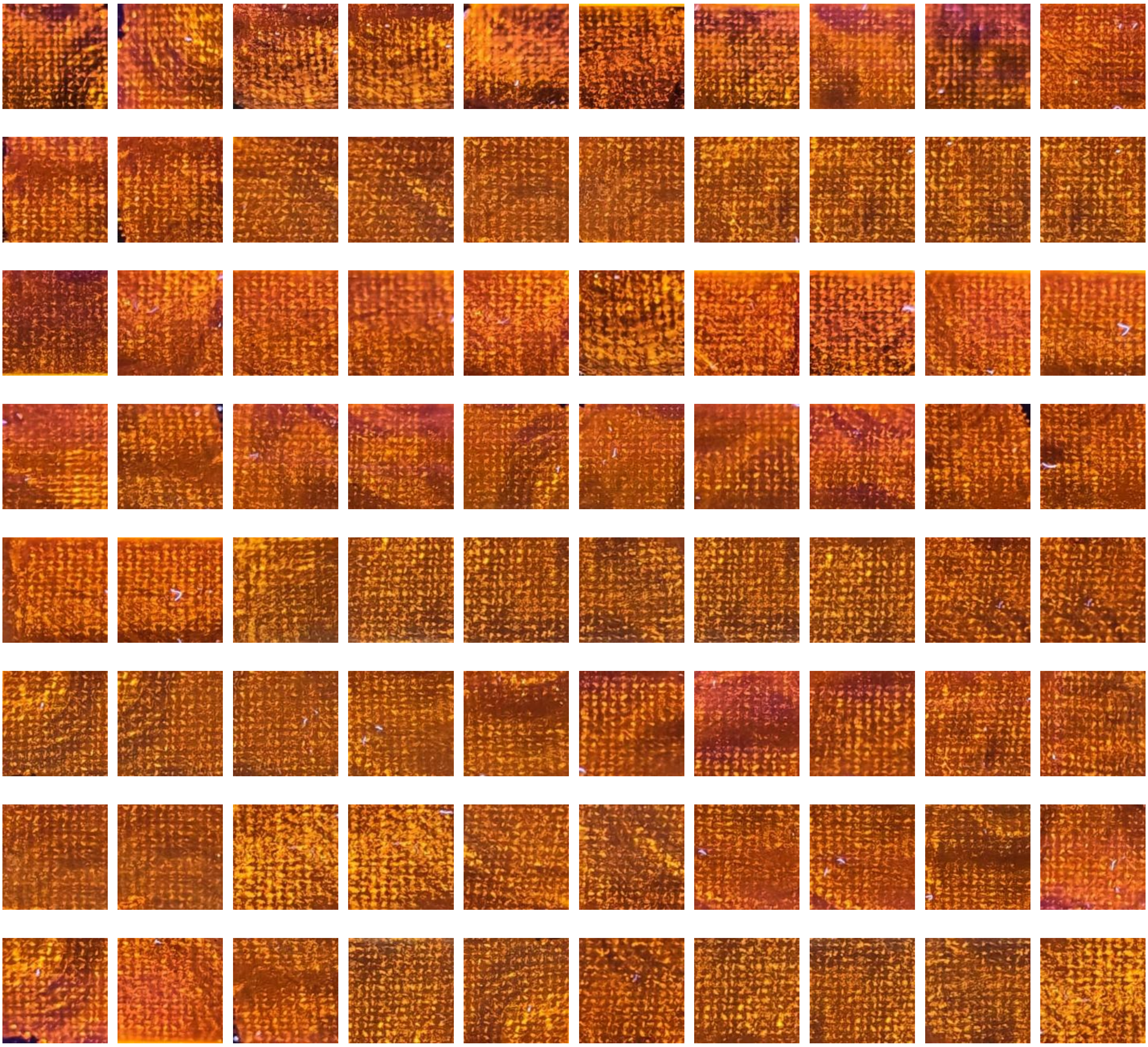


Figure S24. The image set of the surface textures of plaid fabric inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

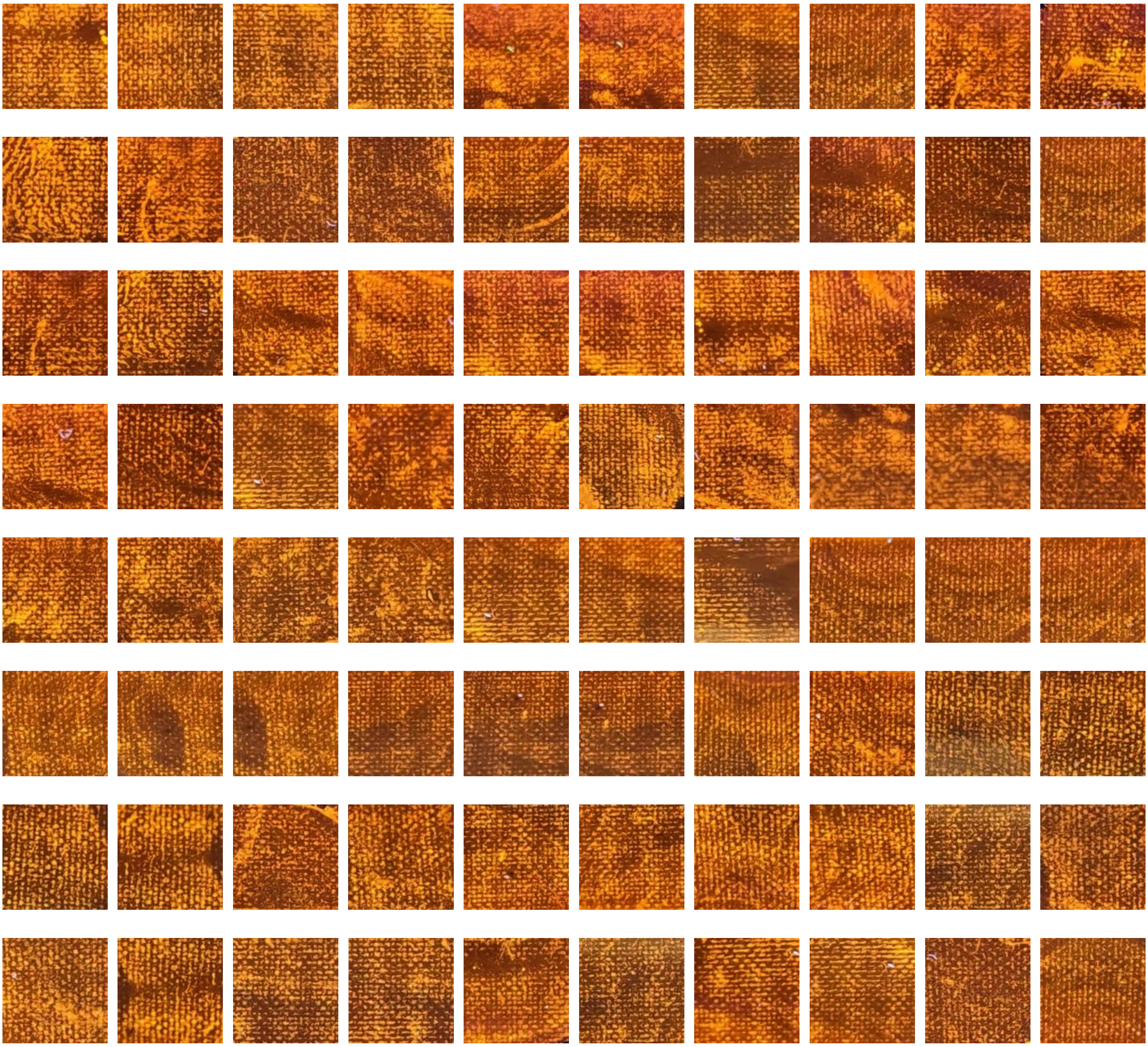


Figure S25. The image set of the surface textures of stripe fabric inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

Figure S26. The image set of the surface textures of rubber inputted into CNN model for training and recognition, containing 100 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

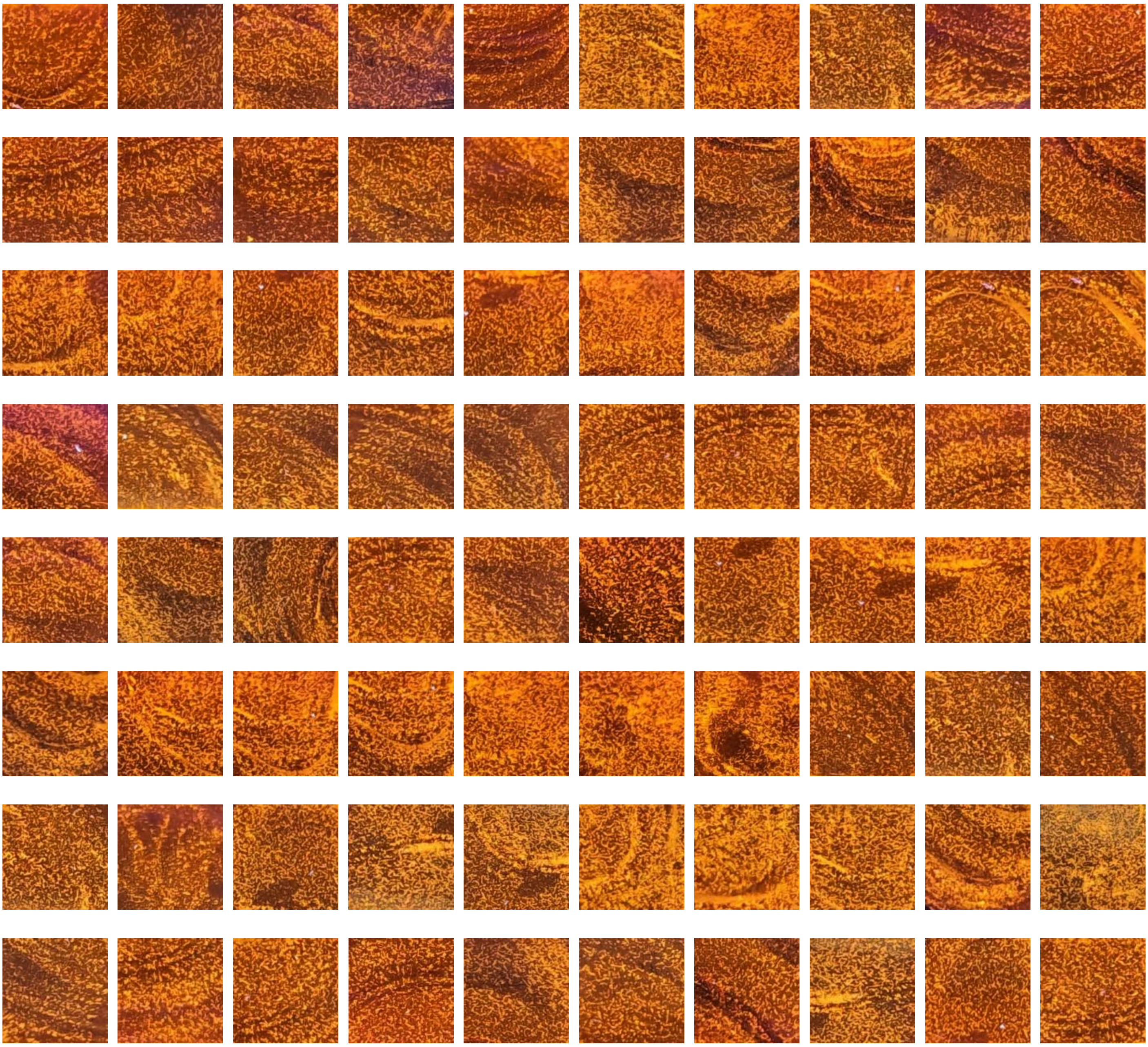


Figure S27. The image set of the surface textures of sponge inputted into CNN model for training and recognition, containing 80 fluorescent patterns recorded by **MNIMP** films. Photographs were taken by a built-in camera of a smartphone in a dark box with 365 nm UV irradiation.

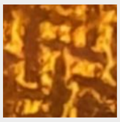
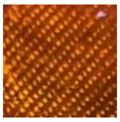
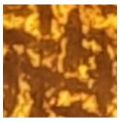
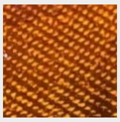
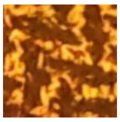
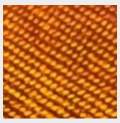
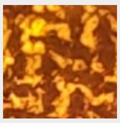
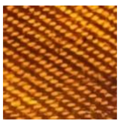
Table S1. Photophysical data of **MNIMP**. λ_{ab}= maximum absorption wavelength, λ_{em} = maximum emission wavelength, τ = fluorescence lifetime, Δ*E* = Energy gap, D_{CT} = charge transfer distance. QY = fluorescence quantum yield

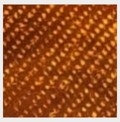
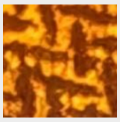
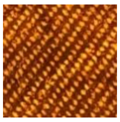
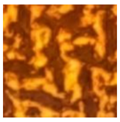
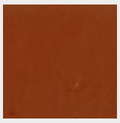
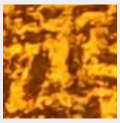
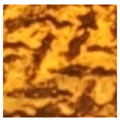
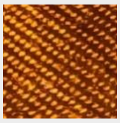
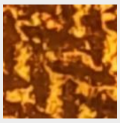
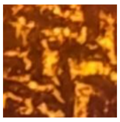
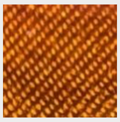
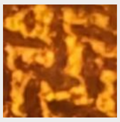
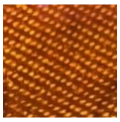
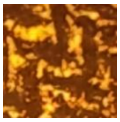
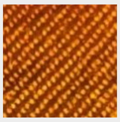
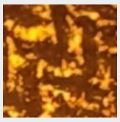
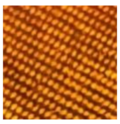
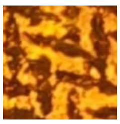
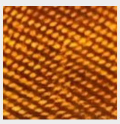
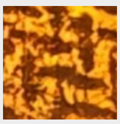
Parameter	**Value**	
λ_{ab}	Hexane	365 nm
	Ethyl acetate	371 nm
	THF	373 nm
	Methanol	374 nm
	DMF	379 nm
λ_{em}	Hexane	549 nm
	Ethyl acetate	560 nm
	THF	563 nm
	Methanol	561 nm
	DMF	572 nm
	Crystalline powder	573 nm
Stokes shift	Hexane	184 nm
	Ethyl acetate	189 nm
	THF	190 nm
	Methanol	187 nm
	DMF	193 nm
τ	Crystalline powder	0.85 ns
QY	THF solution	＜1%
	Solid powder	4.2%
	As-prepared film	＜%
	Self-assembled film	3.3%
Energy levels	HOMO	-6.37 eV
	LUMO	-3.13 eV
Δ*E*	3.24 eV	
D_{CT}	4.24 Å	

Table S2. Basic crystallographic data of **MNIMP** from single crystal X-ray diffraction analysis.

Empirical formula	$C_{28}H_{24}N_4O_8$
Compound name	(*E*)-2-(((2-methoxy-4-nitrophenyl)imino)methyl)phenol
Synonym	MNIMP
Space group	P 2_1/n (14)
Cell lengths	a = 15.9306(2) Å b = 6.83720(10) Å c = 22.9367(3) Å
Cell angles	α = 90° β = 94.7220(10) γ = 90°
Cell Volume	2489.8
Z, Z'	Z: 4 Z': 1
R-Factor (%)	3.63

Table S3. Mean intensity values of background noise, texture pattern and contrast degree obtained from 20 repeated parallel fluorescence imaging experiments.

Group	Background noise	Mean Intensity	Contrast value
1	 65.657	 120.911	 148
2	 60.863	 101.085	 150
3	 65.508	 110.022	 147
4	 68.340	 107.907	 157
5	 71.632	 122.884	 153
6	 66.456	 112.859	 142

7	 65.956	 99.709	 141
8	 66.418	 104.778	 143
9	 69.045	 101.756	 144
10	 66.975	 110.834	 154
11	 66.481	 101.566	 154
12	 63.205	 116.676	 141
13	 63.523	 110.719	 156
14	 67.327	 111.752	 156
15	 66.293	 116.316	 153
16	 68.384	 111.989	 145
17	 60.554	 113.884	 145

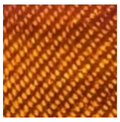
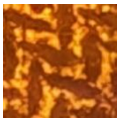
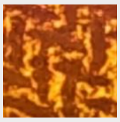
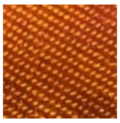
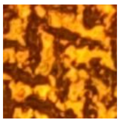
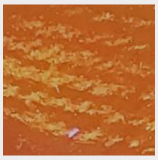
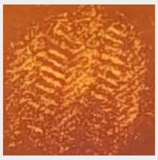
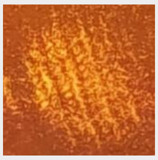
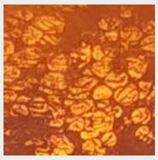
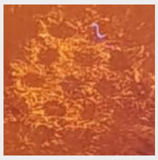
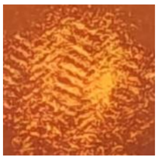
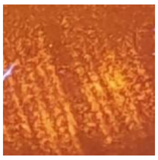
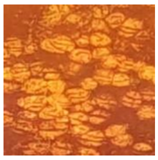
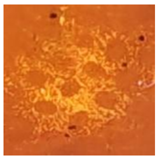
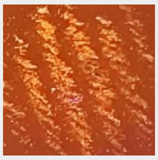
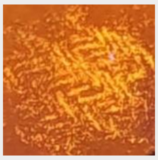
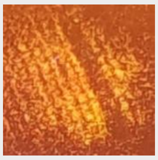
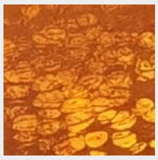
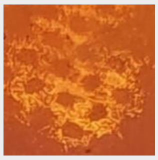
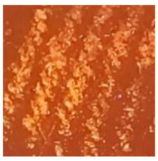
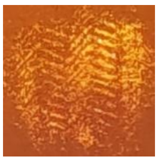
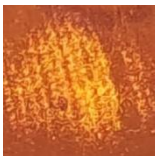
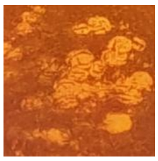
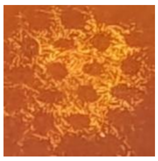
18	 71.939	 106.400	 160
19	 69.376	 109.490	 145
20	 66.278	 105.478	 148

Table S4. The fluorescent images surface textures (twill, gunny, polyester, soft foam, and nonwoven) inputted into CNN model for recognition. Photographs were taken in a dark box with 365 nm UV irradiation.

Angle	Twill	Gunny	Polyester	Soft foam	Nonwoven
15°	 **Twill √ (64.54%)**	 **Sponge × (20.34%)**	 **Polyester √ (20.39%)**	 **Soft foam √ (72.00%)**	 **Nonwoven √ (68.65%)**
30°	 **Twill √ (67.07%)**	 **Sponge × (35.45%)**	 **Polyester √ (55.97%)**	 **Soft foam √ (68.65%)**	 **Nonwoven √ (82.48%)**
45°	 **Twill √ (72.30%)**	 **Plastic × (32.60%)**	 **Polyester √ (44.66%)**	 **Soft foam √ (54.52%)**	 **Nonwoven √ (87.13%)**
60°	 **Twill √ (73.68%)**	 **Gunny √ (34.52%)**	 **Polyester √ (67.80%)**	 **Soft foam √ (68.30%)**	 **Nonwoven √ (94.10%)**

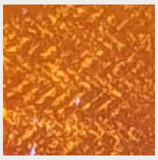
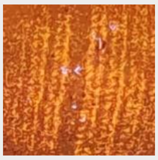
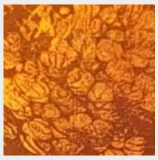
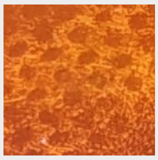
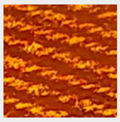
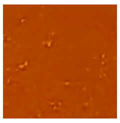
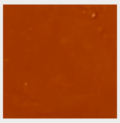
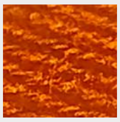
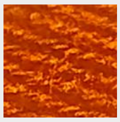
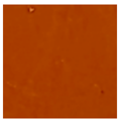
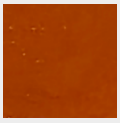
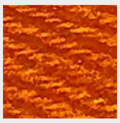
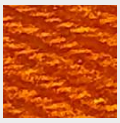
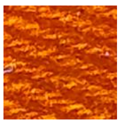
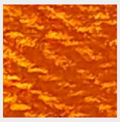
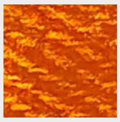
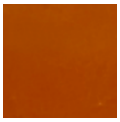
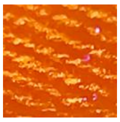
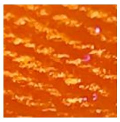
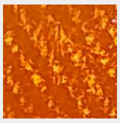
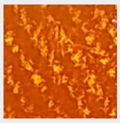
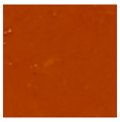
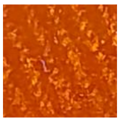
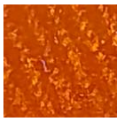
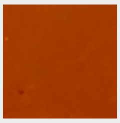
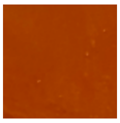
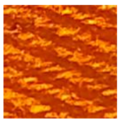
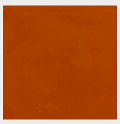
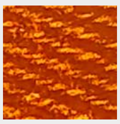
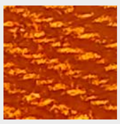
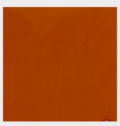
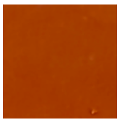
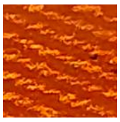
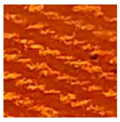
90°	 **Twill √ (88.29%)**	 **Gunny √ (69.51%)**	 **Polyester √ (74.93%)**	 **Soft foam √ (78.65%)**	 **Nonwoven √ (96.16%)**

Table S5. Mean intensity values of background noise, texture pattern and contrast degree obtained from 20 repeated parallel fluorescence imaging experiments under practical operating conditions.

Group	Background noise	Mean intensity	Contrast value
1	 74.101	 104.358	 156
2	 73.623	 109.163	 153
3	 74.385	 103.152	 155
4	 70.694	 103.833	 155
5	 72.272	 105.539	 149
6	 74.161	 105.832	 149
7	 73.22	 101.878	 150
8	 72.054	 102.546	 159

9	 72.545	 100.794	 151
10	 74.494	 100.214	 160
11	 70.955	 100.794	 156
12	 75.429	 102.278	 149
13	 73.33	 100.619	 153
14	 74.586	 107.249	 148
15	 69.614	 100.079	 153
16	 73.425	 106.775	 147
17	 74.108	 107.341	 153
18	 70.928	 100.937	 155
19	 75.113	 104.039	 153

20	 74.566	 104.839	 150

3. Quantum chemical calculation

The geometry optimization of **MNIMP** at ground state was carried out in THF solvent by B3LYP/6-31+G* method. Based on the optimized geometry, the harmonic vibrational frequency calculation was performed at the same theoretical level to check the nature of the stationary point. The electron-hole coherence of charge transfer upon electronic transition was investigated via charge density difference (CDD) calculation in Multiwfn 3.8 software [1]. The charge transfer index (D_{CT}) and the centroids of charge $C_{+}(r)/C_{-}(r)$ for the first excited state (S_1) of **MNIMP** were evaluated at PCM(THF)/TD-B3LYP/6-31+G* level. The resulting plots were obtained from Visual Molecular Dynamics (VMD) software [2]. All calculations were performed in Gaussian 09 program package [3].

4. Introduction of Convolutional Neural Network and Resnet-18 model

Convolutional Neural Network (CNN) is a type of neural network specially designed to deal with data with similar grid structure, such as sequence data (regarded as the data sampled regularly in a certain dimension) and image data, *etc*. Among them, for the processing of large-scale image sets, CNN has excellent performance [4,5]. CNN mainly includes the following structures: input layer, convolutional layer, pooling layer and fully connected layer [6]. Herein, we take the recognition-classification of fluorescent patten images as an example to illustrated the working process of CNN. Firstly, the input layer is responsible for inputting patten images into the model. Since the initial specifications of the images are different and the amount of manually captured data is still limited, a series of data pre-processing operations such as cropping, conversion and data increment operations will be performed at the input layer. The convolutional layer, as the core layer of the CNN, is responsible for the convolution operation on the results of the previous layer. Specifically speaking, the convolution operation is a special type of linear transformation that extracts feature values by sliding convolution kernels over an image matrix. Each convolution kernel can be considered as a filter, or as a small 2D matrix, which overlaps with local regions of the input images and performs a dot product operation to obtain the feature values by summing. After computing all the regions of the input images, a new feature map is obtained to describe the features of the input images. Generally, it requires multiple convolutional layers to obtain more complex feature representations. The pooling layer is another important part of CNN, which is used to aggregate the feature maps from convolution to obtain smaller feature maps. It thus reduces the number of parameters to improve the computational efficiency of the model. Maximum pooling and average pooling are two major methods commonly included in pooling layer. In this work, the former is adopted to select the maximum value from the pooling window for output and compress the size of the feature map. The fully connected layer is the final computational layer in CNN, which connects all the feature maps obtained after previous convolution operations and converts them into one-dimensional vectors, whereby, the classification results are obtained by weighting operations and output.

According to the principle of traditional convolutional neural network, more convolutional layers together with deeper network would lead to complex fitted results. But in fact, blindly deepening the depth of the model may cause shortcomings such as poor fitting effect and disappearing gradient, *ect*. In order to solve these problems and improve the accuracy and efficiency of image classification, a residual network with a depth of 18 layers (Resnet-18) is employed on the basis of traditional CNN [7]. The core idea of Resnet is to add a residual connection, allowing the output of a particular layer to skip multiple layers of computation and be fed directly to subsequent layers (Figure S23). In this case, even if the effect of some convolutional layers is not significant, the output result will still transmit the information of the previous layer to avoid excessive gradient loss.

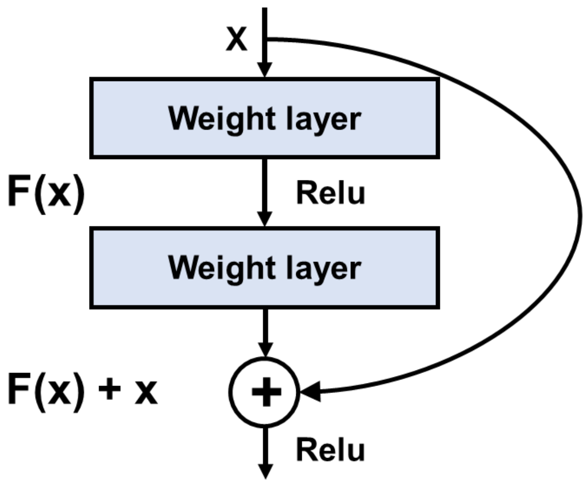


Figure S28. Schematic diagram showing the basic unit structure of Resnet.

Resnet-18 consists of five convolutional layers, one fully connected layer, several pooling layers and Relu (rectified linear unit) functions. The specific parameters of each convolutional layer of ResNet-18 are demonstrated in Table S3. Among them, except the convolution kernel size of 7×7 in Conv1, Conv2-4 are residual units with convolution kernel size of 3×3, and the number of channels is twice that of the upper layer. After the preprocessing of the input layer, the dimension of the input image is unified to 224×224, and the prediction results of 15 classifications will be output in the fully connected layer after the processing of the above five convolutional layers.

Table S6. The structure of Resnet-18.

Layers of Resnet-18	Size of convolution kernels
Conv 1	7×7, 64
Conv 2	[3×3, 64; 3×3, 64] ×2
Conv 3	[3×3, 128; 3×3, 128] ×2
Conv 4	[3×3, 256; 3×3, 256] ×2
Conv 5	[3×3, 512; 3×3, 512] ×2
Fully connected layer	

5. CNN code with modifications

```
# Retrieve the required functional modules directly from the API

import torch

import torchvision.transforms as transforms

import torchvision.datasets as datasets

import torch.nn as nn

import torch.optim as optim

from torchvision.models import resnet18 # Direct access to the open source resnet-18 model code

# Resnet-18 Model code official path: https://download.pytorch.org/models/resnet18-5c106cde.pth
```

```
# Resnet18 model code reference:
  https://github.com/GarsonWw/resnetgarson/blob/master/resnet/model.py
from torch.utils.data import DataLoader, random_split
from torch.utils.tensorboard import SummaryWriter
from torch.optim.lr_scheduler import StepLR
from sklearn.metrics import accuracy_score
from PIL import Image
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
---------------------------------------------------------------------------------------------------------------
# Call device run
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
---------------------------------------------------------------------------------------------------------------
# Define data transformation and enhancement
# Image preprocessing operation strategy reference code:
https://github.com/GarsonWw/resnet-garson/blob/master/resnet/model.py
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
```

```
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
# Loading the dataset
data_dir = "data/train1"   # Dataset path
dataset = datasets.ImageFolder(root=data_dir, transform=train_transform)

# Split the dataset into a training set and a test set, using 80% of the data as the training set
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
# Initialize the ResNet-18 model
```

```
model = resnet18(pretrained=True) # Use the official resnet18 to load the pre-trained model

num_ftrs = model.fc.in_features

model.fc = nn.Linear(num_ftrs, len(dataset.classes))

model = model.to(device)
```

```
# Define loss functions and optimizers

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight_decay=1e-5)


# Set up the learning rate scheduler

scheduler = StepLR(optimizer, step_size=10, gamma=0.1)


# Create a SummaryWriter object

writer = SummaryWriter('logs')
```

```
# Training model

num_epochs = 150

best_acc = 0.0

patience = 5    # Define patience value

no_improve_count = 0    # Record the number of times the validation set accuracy has not
improved

for epoch in range(num_epochs):
```

```
        model.train()
        running_loss = 0.0
        corrects = 0
        total = 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)

            _, preds = torch.max(outputs, 1)
            corrects += torch.sum(preds == labels.data)
            total += labels.size(0)

        epoch_loss = running_loss / len(train_dataset)
        epoch_acc = corrects.double() / total
        print(f"Epoch {epoch + 1}/{num_epochs} Training Loss: {epoch_loss:.4f} Training Acc: {epoch_acc:.4f}")
        scheduler.step()
```

```
# Evaluate the model on a validation set
    model.eval()
    test_loss = 0.0
    test_corrects = 0
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        with torch.no_grad():
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item() * inputs.size(0)
            _, preds = torch.max(outputs, 1)
            test_corrects += torch.sum(preds == labels.data)
    test_loss = test_loss / len(test_dataset)
    test_acc = test_corrects.double() / len(test_dataset)
    print(f"Epoch {epoch + 1}/{num_epochs} Test Loss: {test_loss:.4f} Test Acc:
{test_acc:.4f}")

# Write training and test results to TensorBoard
    writer.add_scalar('Loss/train', epoch_loss, epoch)
    writer.add_scalar('Accuracy/train', epoch_acc, epoch)
    writer.add_scalar('Loss/test', test_loss, epoch)
```

```
        writer.add_scalar('Accuracy/test', test_acc, epoch)
```

```
# Define class tag
classes = ['twill', 'gunny', 'elastic', 'polyester', 'gauze', 'hard froth', 'soft froth', 'leather', 'linen',
'nonwovens', 'plastic', 'plaid fabric', 'stripe fabric', 'rubber', 'sponge']
```

```
# Evaluate the model on a validation set and generate a confusion matrix
model.eval()
true_labels = []
predicted_labels = []
for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)
    with torch.no_grad():
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        true_labels.extend(labels.cpu().numpy())
        predicted_labels.extend(preds.cpu().numpy())

# Generate confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)


# Plot confusion matrix
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.xticks(np.arange(len(classes)) + 0.5, classes, rotation=45)
plt.yticks(np.arange(len(classes)) + 0.5, classes, rotation=0)
plt.show()
```

```
# Calculate accuracy and recall rate
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Calculates the training set's metrics
train_true_labels = []
train_predicted_labels = []
for inputs, labels in train_loader:
    inputs, labels = inputs.to(device), labels.to(device)
    with torch.no_grad():
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        train_true_labels.extend(labels.cpu().numpy())
```

```
            train_predicted_labels.extend(preds.cpu().numpy())


train_accuracy = accuracy_score(train_true_labels, train_predicted_labels)
train_precision = precision_score(train_true_labels, train_predicted_labels,
average='weighted')
train_recall = recall_score(train_true_labels, train_predicted_labels, average='weighted')
```

```
# Calculates the testing set's metrics
test_true_labels = []
test_predicted_labels = []
for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)
    with torch.no_grad():
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        test_true_labels.extend(labels.cpu().numpy())
        test_predicted_labels.extend(preds.cpu().numpy())


test_accuracy = accuracy_score(test_true_labels, test_predicted_labels)
test_precision = precision_score(test_true_labels, test_predicted_labels,
average='weighted')
test_recall = recall_score(test_true_labels, test_predicted_labels, average='weighted')
```

```
--------------------------------------------------------------------------------
# Print pointer
print("training set：")
print(f"acc: {train_accuracy:.4f}")
print(f"precision: {train_precision:.4f}")
print(f"recall: {train_recall:.4f}")
print("testing set：")
print(f"acc: {test_accuracy:.4f}")
print(f"precision: {test_precision:.4f}")
print(f"recall: {test_recall:.4f}")
--------------------------------------------------------------------------------
# Save model
torch.save(model.state_dict(), "resnet18_texture_classification.pth")
print("Model saved")
```

6. References

[1] M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G.A. Petersson, et al. Gaussian 09, Revision D.01; Gaussian, Inc.:Wallingford, CT, 2013.

[2] T. Lu, F.W. Chen, J. Comput. Chem. 33 (2012) 580-592.

[3] W. Humphrey, A. Dalke, K. Schulten, J. Mol. Graph. 14 (1996) 33-38.

[4] Y. Liu, J.H. Xue, D.X. Li , W.D. Zhang, T.K. Chiew, Z.J. Xu, Image recognition based on

lightweight convolutional neural network: Recent advances, Image and Vision Computing 146 (2024) 105037.

[5] X.L. Wei, B.Y. Hu, T.S. Gao, J. Wang, B. Deng, Multi-scale convolutional neural network for texture recognition, Displays 75 (2022) 102324.

[6] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Commun. ACM 60 (2017) 84-90.

[7] K.M. He, X.Y. Zhang, S.Q. Ren, J. Sun, Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition (2016) 770-778.