International Journal of Engineering and Information Systems (IJEAIS)
ISSN: 2643-640X
Vol. 9 Issue 9 September - 2025, Pages: 204-211

Building a compact model of Artificial intelligence for detecting road cracks caused by subsidence

Phuong Thao CAO[1], Quang Huy CHU[2]

[1]Data scientist, France Email: thao.cao@iat.com.vn
[2]Intern, ATCEC company, Vietnam Email: huy.chu@iat.com.vn

Abstract: *Environmental and many other reasons can cause the damage and subsidence of road surfaces. We need to detect signs of damage as soon as possible to warn of accidents and carry out repairs promptly. Therefore, technical methods of road surface factors monitor are very important in the construction industry. Currently, there are many image processing approaches of artificial intelligence to detect cracks but they have encountered the obstacle of needing large datasets. To overcome this feature, our research proposes a model based on Encoder - Decoder of artificial intelligence to detect road cracks. The program can run with small datasets and it has high accuracy.*

Keywords: crack detection, crack segmentation, encoder, decoder, image processing approach, road subsidence

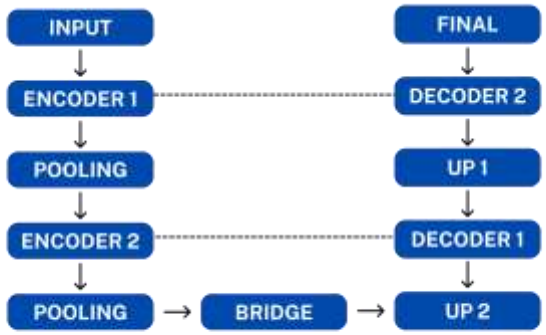
I. INTRODUCTION

In the construction field, to check the quality of construction works, people use traditional methods such as sensors to detect and record vibration indexes. Then they are analysed. If there are abnormalities in the structure of materials and works, there will be an alarm [1][7][8]. We know that there can be many factors that cause the instability to data such as light, weather, equipment durability, human resources, sensor quality,… In construction, maintenance work must be performed regularly and continuously, so the advanced methods need to be researched to reduce the time and labor consumption. Nowadays, people have methods such as Image Processing Approaches using artificial intelligence such as Fully Convolutional Networks (FCN) [3][6]. This is a deep learning network structure proposed for image semantic segmentation tasks. The accuracy is also very promising from 80% to 85%. But to train these models well, large datasets and long training time are needed.

To serve the needs of research and study, people need a model that can optimize costs. In this paper, we build a compact model based on the Encoder - Decoder model [3][6] of artificial intelligence. This program detects cracks in the image, which can be applied to detect cracks on roads due to subsidence.

II. DESIGN OF MODEL FOR CRACK SEGMENTATION - CMFCS [3][6]

The programmed model has a complete pipeline from generating simulation data, training data, and evaluating accuracy. The program can run quickly with high accuracy while only needing to train with a small data set. The CMFCS model is designed with 2 main blocks: Encoder and Decoder. In addition, it includes Pooling, Bridge, and Up sampling steps. The following is a detailed explanation for each block.



CMFCS MODEL

Fig.1. Model

The input is a 128 x 128 Image, then

1. Encoder is responsible for extracting important features of the input image. After performing the Encoder steps from top to bottom, the width x height of the image is reduced, while the depth is increased. The Encoder block has Convolution and ReLU algorithms. For example:

- Conv3×3 is a convolution layer using a 3×3 kernel (filter). It scans through the image or feature map, extracting local features. Padding=1 is often used to keep the spatial dimension (H×W) constant.
- ReLU (Rectified Linear Unit) is a nonlinear activation function with the formula ReLU(x) = max(0, x). It helps the network to learn the nonlinear features and avoid the "vanishing gradient" problem. The encoder applies Conv3×3 → ReLU twice in a row. We use two consecutive filtering layers to learn more complex features.

2. Pooling reduces the image size and retain the important features. For example MaxPool2d(2) means 2×2 kernel and stride = 2. This function takes the maximum value in each 2×2 cell. The output is a reduced spatial size (H, W halved).

3. Bridge does not reduce the size anymore but processes the compressed features. It learns the most abstract features (high-level features). In this program, it is to learn the semantics of crack. Bridge uses the forward propagation algorithm of convolution neural network (CNN) with 2D Convolution and ReLU. Then participates in back propagation during training**.**

4. Up sampling increases the spatial dimension (H×W) of the feature map (opposite to pooling). It combines the detailed information from the encoder (feature map before pooling) with the semantic information from the decoder. This is important because the crack is very thin, if only using up sampling, the detailed edge will be lost. We use ConvTranspose2d to gradually restore the resolution. For example, ConvTranspose2d(2x2, 2) means a 2×2 kernel and a stride = 2. The up sampling steps help the image size to be gradually increased from 32×32 → 64×64 → 128×128.
5. Decoder does the opposite of the Encoder. It gradually increases the size of the image and reconstructs the compressed information in the Encoder. At each stage of the decoder, the corresponding symmetric layer of the encoder is cropped and concatenated.

Final is the final Output, we use:

- Conv2d(1×1) is the convolution with 1×1 kernel. It linearly combines all channels into the desired number of channels. In segmentation binary crack detection, we need 1 output channel (mask). So the Output has the shape (Batch, 1, H, W).
- Sigmoid is an activation function that compresses the value to the interval [0,1]. This means that each pixel has a probability of 0 if it is not cracked. Otherwise, the probability is 1 if it is cracked.

Table 1: Model layers of the CMFCS

STAGE	LAYER TYPE/OPERATION
Encoder	Conv3x3 + ReLU (x2) -> enc1
	MaxPool2d(2) -> pool1
	Conv3x3 + ReLU (x2) -> enc2
	MaxPool2d(2) -> pool2
Bridge	Conv3x3 + ReLU (x2) -> bridge
Decoder	ConvTranspose2d(2x2, stride=2) -> up2
	Concat(skip from enc2)
	Conv3x3 + ReLU (x2) -> dec2
	ConvTranspose2d(2x2, stride=2) -> up1
	Concat(skip from enc1)
	Conv3x3 + ReLU (x2) -> dec1
Output	Conv2d(1x1) -> final
	Sigmoid Activation

III. EXPERIMENT

3.1. Dataset

The simulation dataset consists of 2000 sets (Image x Mask). Each image will have a size of 128×128 pixels, generated by the following algorithm:

- Step 1: Create Image - representing the original image of the road surface, with the background using the Gaussian noise RGB algorithm to create a concave-convex background similar to the real image.
- Step 2: Create Mask which is the image after Crack Segmentation by copying the Image image and converting it to grayscale.
- Step 3: Generate random cracks (points) from top to bottom.
- Step 4: Draw the same points on both Image and Mask. Draw black points on Image to represent black cracks. Draw white points on Mask to represent the cracks detected after segmentation.

Here is the code to create the dataset:

…

```
def create_synthetic_dataset(n_samples=2000, img_size=128, save_dir="data"):
   image_dir = os.path.join(save_dir, "images")
   mask_dir  = os.path.join(save_dir, "masks")
   os.makedirs(image_dir, exist_ok=True)
   os.makedirs(mask_dir, exist_ok=True)

   for i in range(n_samples):
      # --- 1. Gaussian noise ---
      base = np.random.normal(loc=128, scale=30, size=(img_size, img_size, 3)) \
             .clip(0,255).astype(np.uint8)
      img_color = Image.fromarray(base)          # RGB Image
     img_gray  = img_color.convert("L")          #  grey Mask

     # --- 2. Draw random crack ---
      draw_color = ImageDraw.Draw(img_color)      # draw on Image
      draw_mask  = ImageDraw.Draw(img_gray)       # draw on Mask
```

```
    n_cracks = random.randint(1, 5)

    for _ in range(n_cracks):
        x, y = random.randint(0, img_size-1), 0
        points = [(x, y)]
        for step in range(1, img_size, random.randint(5, 15)):
            x += random.randint(-5, 5)
            y = step
            x = max(0, min(img_size-1, x))
            y = max(0, min(img_size-1, y))
            points.append((x, y))

        width = random.randint(1, 3)
        draw_color.line(points, fill=(0,0,0), width=width)   # draw on Image
        draw_mask.line(points, fill=255, width=width)     # draw on Mask

    # --- 3. Save Image and Mask ---
    img_color.save(os.path.join(image_dir, f"img_{i+1:03d}.jpg"))
    img_gray.save(os.path.join(mask_dir, f"mask_{i+1:03d}.png"))

    print(f » There are {n_samples) Image + mask in the directory '{image_dir}' and '{mask_dir}'")
```

The image obtained after running the simulation data generation program is as follows:

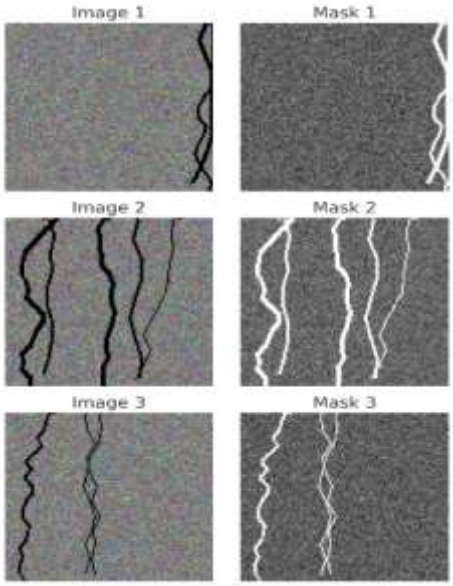


Fig.2. Dataset

3.2. Building CMFCS model

This is the example code for building Encoder, Pooling, Bridge, Decoder and Up sampling such as Figure 1:

```
    # Encoder

    self.enc1 = nn.Sequential(CBR(3,8), CBR(8,8))

    self.pool1 = nn.MaxPool2d(
```

```
        self.enc2 = nn.Sequential(CBR(8,16), CBR(16,16))
        self.pool2 = nn.MaxPool2d(2)

        # Bridge
        self.bridge = nn.Sequential(CBR(16,32), CBR(32,32))

        # Decoder
        self.up2 = nn.ConvTranspose2d(32,16,2,2)
        self.dec2 = nn.Sequential(CBR(32,16), CBR(16,8))
        self.up1 = nn.ConvTranspose2d(8,8,2,2)
        self.dec1 = nn.Sequential(CBR(16,8), CBR(8,8))
        # Final output
        self.final = nn.Conv2d(4,1,1)
    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool1(e1))
        b  = self.bridge(self.pool)
        d2 = self.dec2(torch.cat([self.up2(b), e2], 1))
        d1 = self.dec1(torch.cat([self.up1(d2), e1], 1))
        return torch.sigmoid(self.final(d1))
```

3.3. Training model

In training part, the data is divided 80% for training and 20% for validation. The confusion matrix for pixel-by-pixel segmentation is as follows:

- TP (True Positive): Pixel crack prediction = 1, ground truth = 1.
- TN (True Negative): Pixel background prediction = 0, ground truth = 0.
- FP (False Positive): Pixel prediction = 1 but ground truth = 0.
- FN (False Negative): Pixel prediction = 0 but ground truth = 1.

Total number of pixels = TP + TN + FP + FN.

Example : given an image is 128×128 = 16384 pixels. This image has ground-truth with 1500 crack pixels (number 1 in the mask), the rest is background = 14884. The program predicts as follows:

- There are the 1200 correct crack pixels (TP)
- There are the wrong predicts crack at 300 background pixels (FP)
- The program has missed the 30 crack pixels (FN)
- There are the 14584 correct background pixels (TN)

Table 2: Validation metrics

METRIC	FORMULA	MEANING
IoU	TP / (TP + FP + FN)	Overlap between prediction and ground truth
Dice	2TP / (2TP + FP + FN)	F1-score for segmentation
Precision	TP / (TP + FP)	How many predicted cracks are correct
Recall	TP / (TP + FN)	How many true cracks are detected
F1-score	2*Precision*Recall / (Precision + Recall)	Balance between Precision and Recall

III.4. Evaluation

The program is run experimentally with 2 parts. Part 1 runs experimentally with a dataset of 1000 images and its masks. And the number of epochs is epoch = 5, epoch = 10, epoch = 15, epoch = 20. After running the program, the obtained metrics on the validation dataset are as follows:

Table 3: data of validation

EPOCH NUMBER	IOU	DICE	PRECISION	RECALL	F1-SCORE
5	0.5865	0.7393	0.7055	0.7766	0.7393
10	0.595	0.746	0.7333	0.7592	0.746
15	0.7302	0.8441	0.8452	0.843	0.8441
20	0.8207	0.9015	0.9353	0.8701	0.9015

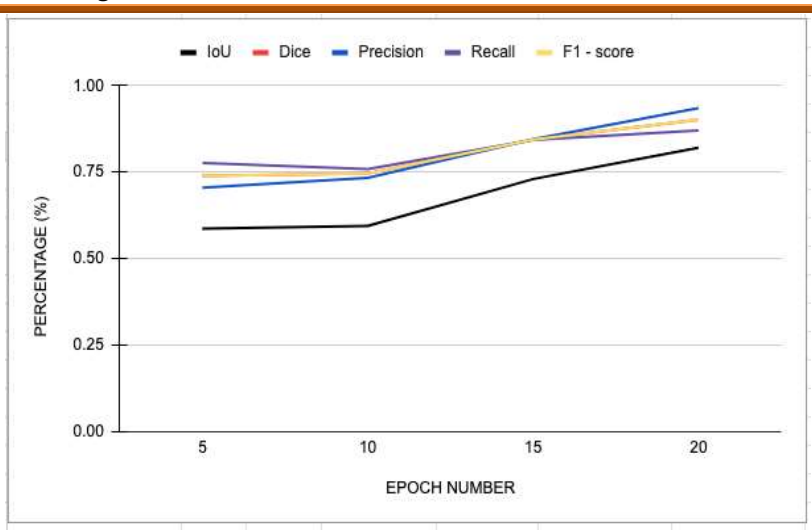


Fig. 3. Chart of validation metrics on dataset of 1000 images

Part 2 runs the experiment with an increased dataset of 200 images and its masks. And the number of epochs is epoch = 5, epoch = 10, epoch = 15, epoch = 20 respectively. After running the program, the metrics data on the validation dataset is obtained as follows:

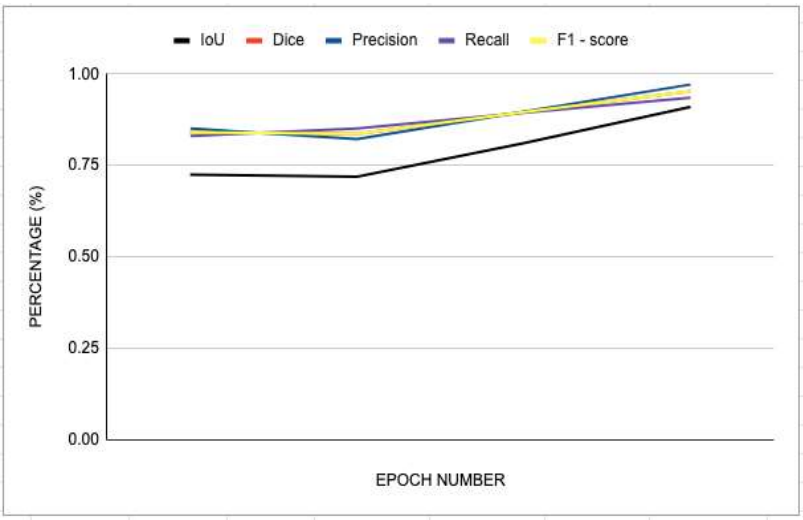


Fig. 4. Chart of validation metrics on dataset of 2000 images

During training, we found that the program will not guarantee accurate results if the dataset size is too small to train. Moreover, if the dataset is small and data augmentation must be used, the spatial location on the image must be preserved so as not to lose spatial information. We also had to experiment with many parameters of the loss function and different learning rates to get the optimal parameter set.

The images used for training are simulated concrete surface data, without images of grass, moss, or other objects. If there are other objects mixed into the images, the training model needs to be built more complex.

After training and evaluation, the accuracy of the program improved when the data size increased from 1000 images to 2000 images. Precision increased from 0.93 to 0.97. This accuracy meets the proposed requirement of detecting cracks on smooth concrete surfaces.

IV. CONCLUSION AND PERSPECTIVE

The research introduces a program used to identify cracks on concrete surfaces. With a compact model, it allows training with a small data set but still has high accuracy. The input is an image simulating a road surface with black cracks. After going through the Encoder and Decode blocks to learn the most characteristic properties, the output is a black and white image. This image has a black dot as the roadbed, a white dot as the crack.

The program continues to be improved to recognize cracks when there are the information noise such as the additional patterns on the image: black dry branches, black moss stains, etc. The program can be upgraded to recognize surfaces such as ceilings, road and bridge surfaces.

V. REFERENCES

1. Dorafshan, S., Maguire, M., & Coopmans, C. (2018). Crack detection using unmanned aerial systems and convolutional neural networks. *Construction and Building Materials, 158*, 312–327. https://doi.org/10.1016/j.conbuildmat.2017.10.105
2. Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). Road crack detection using deep convolutional neural network. *2016 IEEE International Conference on Image Processing (ICIP)*, 3708–3712. https://doi.org/10.1109/ICIP.2016.7533052
3. Cha, Y. J., Choi, W., & Büyüköztürk, O. (2017). Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering, 32*(5), 361–378. https://doi.org/10.1111/mice.12263
4. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 234–241. https://doi.org/10.1007/978-3-319-24574-4_28
5. Li, S., Zhao, X., & Zhou, G. (2020). Automatic crack detection and measurement using deep convolutional neural networks. *Measurement, 152*, 107377. https://doi.org/10.1016/j.measurement.2019.107377
6. Amhaz, R., Chambon, S., Idier, J., & Baltazart, V. (2016). Automatic crack detection on two-dimensional pavement images: An algorithm based on minimal path selection. *IEEE Transactions on Intelligent Transportation Systems, 17*(10), 2718–2729. https://doi.org/10.1109/TITS.2016.2524308
7. Dung, C. V., & Anh, L. D. (2019). Autonomous concrete crack detection using deep fully convolutional neural network. *Automation in Construction, 99*, 52–58. https://doi.org/10.1016/j.autcon.2018.11.028
8. Zou, Q., Zhang, Z., Li, Q., Qi, X., Wang, Q., & Wang, S. (2019). DeepCrack: Learning hierarchical convolutional features for crack detection. *IEEE Transactions on Image Processing, 28*(3), 1498–1512. https://doi.org/10.1109/TIP.2018.2878966
9. Maeda, H., Sekimoto, Y., Seto, T., Kashiyama, T., & Omata, H. (2018). Road damage detection using deep neural networks with images captured through a smartphone. *arXiv preprint arXiv:1801.09454*.
10. Oliveira, H., & Correia, P. L. (2013). Automatic road crack segmentation using entropy and image dynamic thresholding. *EUSIPCO 2013 Proceedings*, 2375–2379.
11. Yang, X., Li, H., Yu, Y., Luo, X., Huang, T., & Yang, X. (2018). Automatic pixel-level crack detection and measurement using fully convolutional network. *Computer-Aided Civil and Infrastructure Engineering, 33*(12), 1090–1109. https://doi.org/10.1111/mice.12387
12. Shi, Y., Cui, L., Qi, Z., Meng, F., & Chen, Z. (2016). Automatic road crack detection using random structured forests. *IEEE Transactions on Intelligent Transportation Systems, 17*(12), 3434–3445. https://doi.org/10.1109/TITS.2016.255224
13. Bang, S., Park, S., Kim, H., & Kim, H. (2019). Encoder–decoder network for pixel-level road crack detection in black-box images. *Computer-Aided Civil and Infrastructure Engineering, 34*(8), 713–727. https://doi.org/10.1111/mice.12436
14. Nguyen, T. T., Nguyen, T. L., Pham, H. H., & Hoang, V. (2021). Crack detection in asphalt pavement images using deep learning and edge detection. *Applied Sciences, 11*(3), 1234. https://doi.org/10.3390/app11031234
15. Xu, Y., Li, Y., Wang, Y., & Li, X. (2019). Structural health monitoring and crack detection using convolutional neural networks. *Smart Materials and Structures, 28*(4), 045019. https://doi.org/10.1088/1361-665X/ab0c10