ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN
MÔN: NHẬP MÔN THỊ GIÁC MÁY TÍNH

ĐỀ TÀI
PHÁT HIỆN VẾT NỨT TRÊN TƯỜNG BÊ TÔNG

Giảng viên hướng dẫn: TS. Mai Tiến Dũng

Sinh viên thực hiện:

Họ tên: Triệu Tấn Đạt

Mã số sinh viên: 22520240

TP. Hồ Chí Minh, tháng 6, năm 2023

Mục lục

I. Lý do chọn đề tài

- Hiện nay, khi thế giới ngày càng phát triển, các công trình được xây dựng ngày càng nhiều. Bên cạnh những lợi ích mà công trình đó mang lại thì vẫn còn tồn tại những nguy hiểm tiềm ẩn. Những vết nứt có thể sẽ hình thành khi công trình được đưa vào sử dụng lâu dài, những vết nứt này sẽ phát triển một cách âm thầm dẫn đến công trình bị sụp đổ. Vì vậy cần phát hiện những vết nứt trên bề mặt tường bê tông càng sớm càng tốt, dưới đây là một số lý do:

+ Bảo đảm an toàn kết cấu: Vết nứt có thể là dấu hiệu của sự suy yếu hoặc hỏng hóc trong kết cấu. Nếu không được phát hiện và xử lý kịp thời, các vết nứt này có thể lan rộng và gây ra các vấn đề nghiêm trọng hơn, thậm chí dẫn đến sự sụp đổ của công trình

+ Ngăn ngừa thấm nước: Vết nứt trên tường có thể tạo điều kiện cho nước xâm nhập vào bên trong, gây ra ẩm mốc và làm hỏng các vật liệu nội thất, đồng thời có thể làm ăn mòn cốt thép bên trong bê tông. Phát hiện sớm vết nứt giúp ngăn ngừa các vấn đề về thấm nước và ẩm mốc.

+ Bảo vệ giá trị tài sản: Công trình có vết nứt có thể bị giảm giá trị trên thị trường bất động sản. Việc phát hiện và sửa chữa kịp thời các vết nứt giúp duy trì và nâng cao giá trị của tài sản.

+ Giảm chi phí sửa chữa: Phát hiện vết nứt sớm giúp tránh các hư hỏng lớn hơn và phức tạp hơn, từ đó giảm chi phí sửa chữa so với việc phải khắc phục các vấn đề nghiêm trọng hơn sau này.

+ Đánh giá nguyên nhân gốc rễ: Việc phát hiện vết nứt giúp các chuyên gia xây dựng xác định nguyên nhân gốc rễ của vấn đề, chẳng hạn như lún móng, sự co ngót của bê tông, hoặc tải trọng quá lớn. Từ đó, họ có thể đề ra giải pháp khắc phục phù hợp và hiệu quả.

+ Duy trì thẩm mỹ của công trình: Vết nứt trên tường không chỉ ảnh hưởng đến cấu trúc mà còn làm giảm tính thẩm mỹ của công trình. Phát hiện và sửa chữa vết nứt giúp duy trì vẻ đẹp và sự hoàn thiện của công trình.

+ Bảo vệ sức khỏe người sử dụng: Vết nứt có thể tạo điều kiện cho vi khuẩn và nấm mốc phát triển, gây ra các vấn đề về sức khỏe cho người sử dụng công trình. Việc phát hiện sớm và khắc phục vết nứt giúp bảo vệ môi trường sống lành mạnh.

II. Phát biểu bài toán

Áp dụng phương pháp máy học để giải quyết bài toán:

- Về thuật toán: Hồi quy Logistic (Logistic Regression)

- Về đặc trưng: sử dụng bộ lọc Sobel để tìm ra cạnh

- Về dữ liệu:

+ Đầu vào:

- Dataset: gồm 40000 ảnh được phân vào 2 lớp Positive và Negative theo tỉ lệ 1:1. Tập dataset sẽ được xử lí trước rồi chia thành 3 tập con: training set, validation set, test set theo tỉ lệ 6:2:2
- Ảnh số: bức ảnh chụp tường bê tông

+ Đầu ra:

- Nhãn của ảnh số: 0 đại diện cho Negative hay ảnh không có vết nứt, 1 đại diện cho Positove hay ảnh có vết nứt

III. Phương pháp

3.1 Giới thiệu

- Hồi quy Logistic là một mô hình thống kê được sử dụng để phân loại nhị phân, tức dự đoán một đối tượng thuộc vào một trong hai nhóm. Hồi quy Logistic làm việc dựa trên nguyên tắc của hàm sigmoid – một hàm phi tuyến tự chuyển đầu vào của nó thành xác suất thuộc về một trong hai lớp nhị phân.

- Sự phát triển của thuật toán này là kết quả của nhiều đóng góp từ các nhà khoa học khác nhau trong suốt nhiều thế kỷ.

3.2 Ý tưởng chính

- Ý tưởng của hồi quy Logistic khá giống với hồi quy tuyến tính. Tuy nhiên cũng có một số khác biệt:

+ Về mô hình:

- Hồi quy Logistic: $f(x) = \frac{1}{1+e^{-(wx+b)}}$
- Hồi quy tuyến tính: $f(x) = wx + b$

+ Về đầu ra:

- Hồi quy Logistic: giá trị thuộc đoạn [0,1]
- Hồi quy tuyến tính: giá trị thực bất kì

+ Hàm mất mát:

- Hồi quy Logistic: $L(w) = \sum_{i=1}^{N} -(y_i \log(a_i) + (1-y_i)\log(1-a_i))$ (Negative Log Likelihood)
- Hồi quy tuyến tính: $L(w) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \mathbf{x_i}w)^2$ (Mean Squared Error)

(Chú thích: N là số điểm dữ liệu, $a_i = \frac{1}{1+e^{-(w\mathbf{x_i}+b)}}$)

3.3 Cài đặt

- Dùng Gradient Descent để tìm ra bộ tham số tối ưu cho mô hình

- Định nghĩa một số hàm liên quan:

+ Hàm mất mát trên 1 điểm dữ liệu:

```
def loss(y, a): # a = sigmoid(z)
    return -1 * (y * np.log(a) + (1-y) * np.log(1-a))
```

(Với y là nhãn thật của dữ liệu, a là nhãn do mô hình dự đoán)

+ Hàm sigmoid:

```
def sigmoid(z): # z = wx + b
    return 1/(1 + np.exp(-z))
```

(Với z = wx + b)

- Cài đặt:

```
def Logistic_Regression(X_train, y_train, learning_rate, epoch):
    # Khởi tạo w, b
    w = np.zeros(X_train.shape[1])
```

```
b = 0.0

for epoch in range(epoch):
    dw = np.zeros(w.shape)
    db = 0.0
    total_loss = 0.0

    for i in range(X_train.shape[0]):
        # Lấy ra dữ liệu thứ i
        x_i = X_train[i,:]
        y_i = y_train[i]

        # Trước khi tính Loss
        z_i = w.dot(x_i) + b
        a_i = sigmoid(z_i)

        # Loss của 1 điểm dữ liệu
        loss_i = loss(y_i, a_i)

        # Đạo hàm
        dw_i = x_i * (a_i - y_i) # dL / dw_i
        db_i = a_i - y_i  # dL / db

        # Tổng đạo hàm, Loss
        dw += dw_i
        db += db_i
        total_loss += loss_i

    # Tính trung bình dw, db, total_loss
    dw = (1.0/X_train.shape[0]) * dw
    db = (1.0/X_train.shape[0]) * db
    total_loss = (1.0/X_train.shape[0]) * total_loss
    print(f'Epoch {epoch+1}, Loss = {total_loss:.2f}')

    # Gradient Descent
    w = w - learning_rate * dw
    b = b - learning_rate * db

    # Dữ liệu dùng để vẽ
    w_cache.append(w.copy())
    b_cache.append(b)
    l_cache.append(total_loss)

print('\nAfter training, we obtained')
print('Weights w = ', w)
```

```
    print(f'Bias b = {b:.2f}')
    return w, b
```

IV. Dữ liệu

4.1 Tìm dữ liệu

- Dữ liệu được tải xuống từ Kaggle

- Dữ liệu gồm 40000 ảnh được chia thành 2 thư mục: Negative (không có vết nứt) và Positive (có vết nứt). Mỗi hình ảnh có kích thước 227x227 pixel.

4.2 Tiền xử lý dữ liệu

4.2.1 Đọc ảnh dưới dạng ảnh xám

- Mục đích:

+ Giảm kích thước của dữ liệu:

- Ảnh màu thường được biểu diễn bằng ba kênh màu (Red, Green, Blue), mỗi kênh cần một giá trị để lưu trữ thông tin về độ sáng của màu đó. Điều này có nghĩa là mỗi điểm ảnh trong một ảnh màu cần ba giá trị.
- Ảnh xám chỉ cần một giá trị để biểu diễn độ sáng, do đó, chuyển đổi ảnh màu sang ảnh xám giúp giảm bớt lượng dữ liệu cần xử lý và lưu trữ.

+ Giảm độ phức tạp của xử lý:

- Làm việc với ảnh xám giúp đơn giản hóa các thuật toán xử lý ảnh. Nhiều phương pháp phân tích hình ảnh (như phát hiện cạnh, phân vùng ảnh, nhận diện hình dạng) hoạt động hiệu quả hơn trên ảnh xám vì chúng chỉ cần xử lý một kênh thay vì ba kênh.
- Điều này giúp giảm thiểu thời gian và tài nguyên tính toán cần thiết cho các tác vụ xử lý ảnh.

+ Thông tin màu sắc không cần thiết:

- Bằng cách sử dụng ảnh xám, có thể tập trung vào các đặc trưng quan trọng hơn mà không bị phân tâm bởi thông tin màu sắc không cần thiết.

- Cách thực thi: `img = cv2.imread(os.path.join(path, img_name), cv2.IMREAD_GRAYSCALE)`

- Giải thích các tham số:

+ `os.path.join(path, img_name)`: đường dẫn tới ảnh

+ `cv2.IMREAD_GRAYSCALE`: chế độ đọc ảnh

4.2.2 Chuyển ảnh về kích thước 120x120

- Mục đích:

+ Đồng nhất kích thước đầu vào:

- Mô hình học máy thường yêu cầu đầu vào có kích thước nhất quán. Việc chuẩn hóa kích thước ảnh đảm bảo rằng tất cả các ảnh đều có cùng số lượng điểm ảnh, giúp đơn giản hóa quá trình xử lý và huấn luyện mô hình.

+ Giảm bớt tài nguyên tính toán

- Ảnh lớn đòi hỏi nhiều tài nguyên (bộ nhớ, thời gian tính toán) hơn để xử lý. Bằng cách giảm kích thước ảnh, có thể tiết kiệm tài nguyên và tăng tốc độ xử lý.

- Cách thực thi: `resized_img = cv2.resize(img, (120, 120))`

- Giải thích các tham số:

+ img: ảnh xám

+ (120, 120): kích cỡ resize

4.2.3 Làm mờ ảnh

- Mục đích:

+ Giúp giảm nhiễu

- Làm mờ ảnh giúp làm giảm nhiễu ngẫu nhiên trong ảnh. Điều này giúp làm sạch ảnh và cải thiện chất lượng của các đặc trưng được trích xuất từ ảnh.

+ Làm nổi bật các đặc trưng quan trọng

- Khi làm mờ ảnh, các chi tiết nhỏ và không cần thiết có thể bị loại bỏ, giúp mô hình tập trung vào các đặc trưng lớn hơn và quan trọng hơn. Điều này có thể giúp mô hình học máy tránh việc học quá mức vào các chi tiết không quan trọng (overfitting).

- Cách thực thi: blured_img = cv2.GaussianBlur(resized_img, ksize=(9,9), sigmaX=0, sigmaY=0)

- Giải thích các tham số:

+ resized_img: ảnh sau khi resize

+ ksize: Đây là kích thước của kernel (hoặc ma trận lọc). Kích thước này phải là một số lẻ và dương. Nó chỉ ra chiều rộng và chiều cao của kernel. Một kernel lớn hơn sẽ tạo ra hiệu ứng làm mờ mạnh hơn nhưng cũng có thể làm mất chi tiết.

+ sigmaX: Đây là độ lệch chuẩn của hàm Gaussian theo trục X. Khi sigmaX được đặt là 0, OpenCV sẽ tự động tính toán giá trị tối ưu dựa trên kích thước của hạt nhân. Độ lệch chuẩn xác định mức độ rộng của hàm Gaussian. Giá trị lớn hơn của sigmaX sẽ làm mờ ảnh nhiều hơn.

+ sigmaY: Đây là độ lệch chuẩn của hàm Gaussian theo trục Y. Tương tự như sigmaX, khi sigmaY được đặt là 0, OpenCV sẽ tự động tính toán giá trị tối ưu. Nếu sigmaY được đặt là 0, giá trị sigmaX sẽ được sử dụng cho cả hai trục, đảm bảo tính đồng nhất của hạt nhân Gaussian theo cả hai trục.

4.2.4 Áp dụng bộ lọc Sobel

- Mục đích:

+ Phát hiện ra cạnh

- Làm nổi bật các khu vực trong ảnh có sự thay đổi độ sáng lớn, tức là các cạnh của các đối tượng trong ảnh

- Cách thực thi:

```
def sobel_filters(img):
    kernels = np.array([[-1,0,1],
                        [-2,0,2],
                        [-1,0,1]])
    return cv2.filter2D(img, -1, kernels)

sobeled_img = sobel_filters(blured_img)
```

4.2.5 Áp dụng threshold

- Mục đích:

+ Làm nổi bật các đặc điểm quan trọng của hình ảnh

- Thresholding giúp phân biệt các phần quan trọng của ảnh (ví dụ, các đối tượng) với phần nền bằng cách chuyển đổi ảnh thành dạng nhị phân (đen trắng). Các pixel có giá trị lớn hơn một ngưỡng cụ thể được đặt thành giá trị tối đa (thường là trắng), và các pixel có giá trị nhỏ hơn ngưỡng được đặt thành giá trị tối thiểu (thường là đen).

- Cách thực thi: `(_, thresholded_img) = cv2.threshold(sobeled_img, 30, 255, cv2.THRESH_BINARY)`

- Giải thích các tham số:

+ sobeled_img: hình ảnh qua bộ lọc Sobel

+ 30: ngưỡng chuyển đổi, những pixel có giá trị lớn hơn 30 sẽ được gán thành 255, những pixel có giá trị nhỏ hơn hoặc bằng 30 sẽ được gán thành 0.

+ cv2.THRESH_BINARY: xác định kiểu ngưỡng. Khi áp dụng ngưỡng nhị phân mỗi pixel của ảnh sẽ được chuyển đổi thành một trong hai giá trị: 0 hoặc 255.

4.2.6 Đưa về dạng vector và lưu dữ liệu

- Cách thực hiện:

```
img_data.append(np.reshape(thresholded_img, -1))
```

```
img_label.append(y_label)
```

4.2.7 Min-max Scaler

- Mục tiêu:

+ Chuyển đổi các giá trị của biến thành một phạm vi cụ thể, thường là từ 0 đến 1.

+ Giúp cân bằng dữ liệu và đảm bảo rằng các biến có cùng phạm vi, từ đó cải thiện hiệu suất của các mô hình học máy.

- Cách thực hiện:

```
min_max_scaler = preprocessing.MinMaxScaler()
img_data = min_max_scaler.fit_transform(img_data)
```

V. Độ đo

- Độ đo Accuracy: là một trong những cách đơn giản và phổ biến nhất để đánh giá hiệu quả của một mô hình học máy, đặc biệt trong các bài toán phân loại. Accuracy thể hiện tỷ lệ phần trăm số lượng dự đoán đúng trên tổng số dự đoán.

- Công thức: $Accuracy = \frac{Số\ lượng\ dự\ đoán\ chính\ xác}{Tổng\ số\ lượng\ dự\ đoán}$

VI. Kết quả

- Mô hình được huấn luyện với learning_rate = 0.1 và epoch = 700.

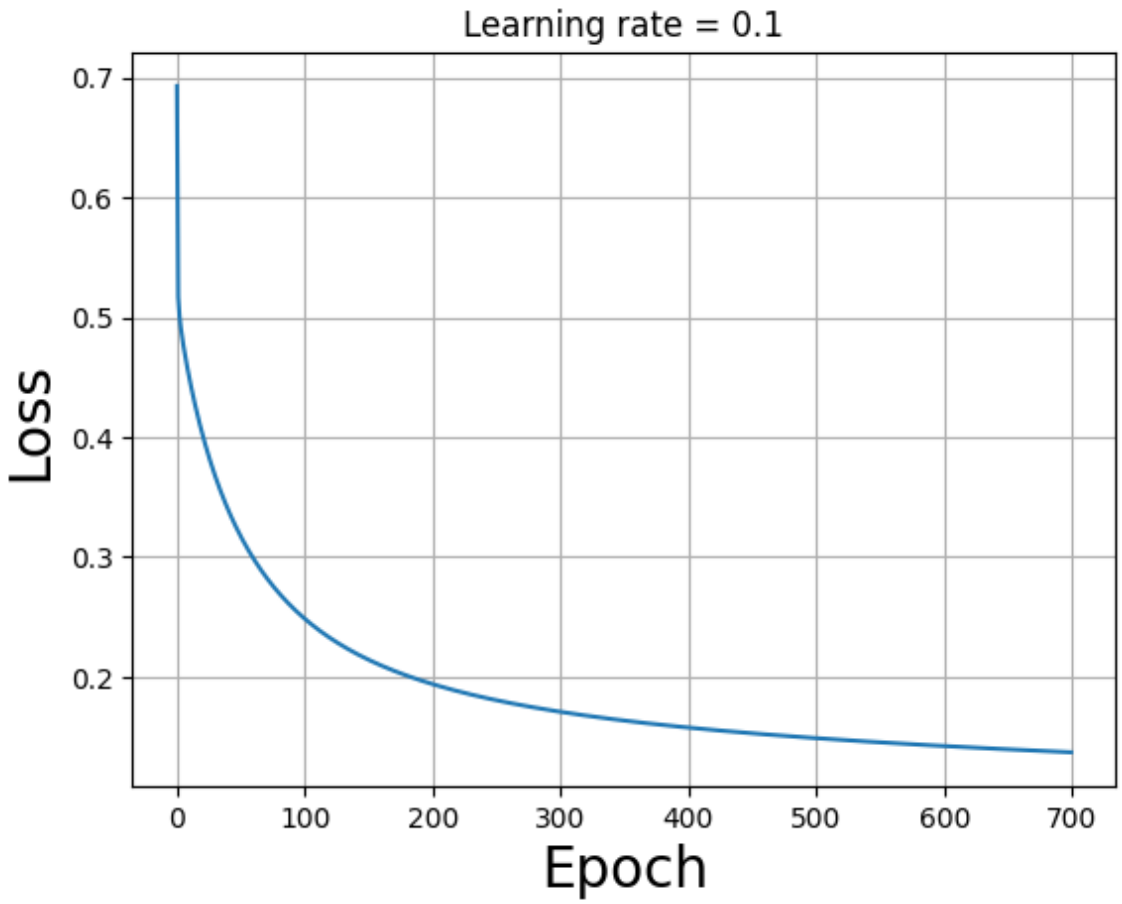
+ Kết quả training:

```
w, b = Logistic_Regression(X_train, y_train, learning_rate = 0.1, epoch = 700)
```

[183] ✓ 29m 13.3s

```
Epoch 1, Loss = 0.69
Epoch 2, Loss = 0.52
Epoch 3, Loss = 0.50
Epoch 4, Loss = 0.49
Epoch 5, Loss = 0.48
Epoch 6, Loss = 0.48
Epoch 7, Loss = 0.47
Epoch 8, Loss = 0.46
Epoch 9, Loss = 0.46
Epoch 10, Loss = 0.45
Epoch 11, Loss = 0.45
Epoch 12, Loss = 0.44
Epoch 13, Loss = 0.44
Epoch 14, Loss = 0.43
Epoch 15, Loss = 0.43
Epoch 16, Loss = 0.42
Epoch 17, Loss = 0.42
Epoch 18, Loss = 0.41
Epoch 19, Loss = 0.41
Epoch 20, Loss = 0.41
Epoch 21, Loss = 0.40
Epoch 22, Loss = 0.40
Epoch 23, Loss = 0.39
Epoch 24, Loss = 0.39
Epoch 25, Loss = 0.39
...
After training, we obtained
Weights w =  [ 0.          0.01282791 -0.01876625 ... -0.04560695 -0.0126589
  0.        ]
Bias b = -3.14
```

+ Giá trị Loss sau mỗi epoch:



+ Đánh giá mô hình:

Evaluate Model

```
p_vali = predict(X_vali, w, b)
p_test = predict(X_test, w , b)
accur_vali = accuracy(p_vali, y_valid)
accur_test = accuracy(p_test, y_test)
print(f'Accuracy on Validation set = {accur_vali:.4f}')
print(f'Accuracy on Test set = {accur_test:.4f}')
```

[186] ✓ 0.5s

```
Accuracy on Validation set = 0.9414
Accuracy on Test set = 0.9380
```

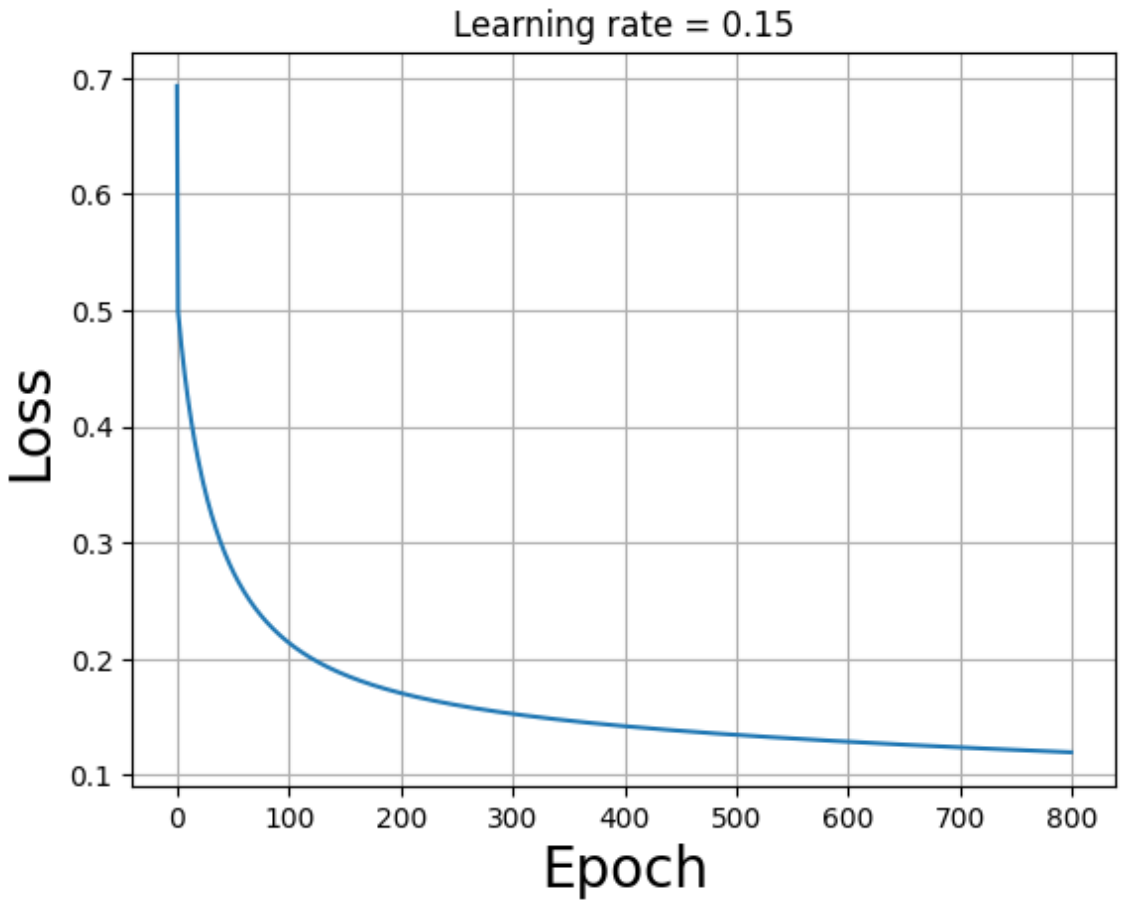
- Mô hình được huấn luyện với learning_rate = 0.15 và epoch = 800

+ Kết quả training:

```
[164]  w, b = Logistic_Regression(X_train, y_train, learning_rate = 0.15, epoch = 800)
       ✓ 33m 40.7s

...    Epoch 1, Loss = 0.69
       Epoch 2, Loss = 0.50
       Epoch 3, Loss = 0.49
       Epoch 4, Loss = 0.48
       Epoch 5, Loss = 0.47
       Epoch 6, Loss = 0.46
       Epoch 7, Loss = 0.45
       Epoch 8, Loss = 0.44
       Epoch 9, Loss = 0.44
       Epoch 10, Loss = 0.43
       Epoch 11, Loss = 0.42
       Epoch 12, Loss = 0.42
       Epoch 13, Loss = 0.41
       Epoch 14, Loss = 0.40
       Epoch 15, Loss = 0.40
       Epoch 16, Loss = 0.39
       Epoch 17, Loss = 0.39
       Epoch 18, Loss = 0.38
       Epoch 19, Loss = 0.38
       Epoch 20, Loss = 0.37
       Epoch 21, Loss = 0.37
       Epoch 22, Loss = 0.36
       Epoch 23, Loss = 0.36
       Epoch 24, Loss = 0.35
       Epoch 25, Loss = 0.35
       ...
       After training, we obtained
       Weights w =  [ 0.          0.0189588  -0.0258831  ... -0.06750773 -0.02031561
         0.        ]
       Bias b = -3.56
```

+ Giá trị Loss sau mỗi epoch:



+ Đánh giá mô hình:

Evaluate Model

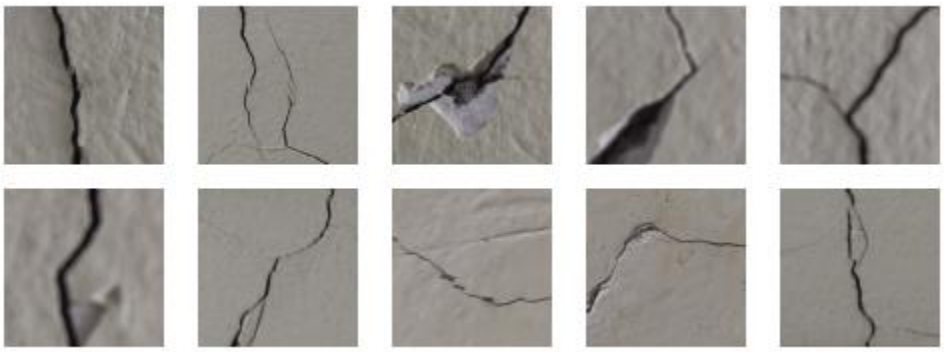
```
p_vali = predict(X_vali, w, b)
p_test = predict(X_test, w , b)
accur_vali = accuracy(p_vali, y_valid)
accur_test = accuracy(p_test, y_test)
print(f'Accuracy on Validation set = {accur_vali:.4f}')
print(f'Accuracy on Test set = {accur_test:.4f}')
```

[167] ✓ 1.4s

```
Accuracy on Validation set = 0.9414
Accuracy on Test set = 0.9397
```

VII. Minh họa

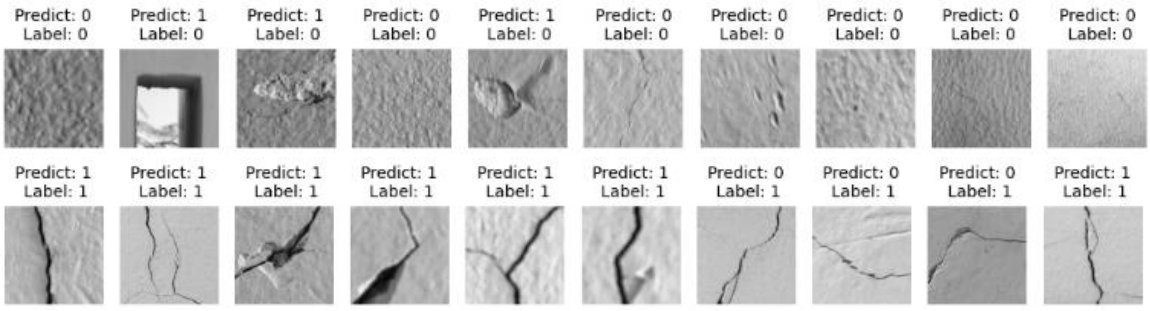
- Tập test được chuẩn bị gồm 10 ảnh có nhãn là Positive và 10 ảnh có nhãn là Negative.



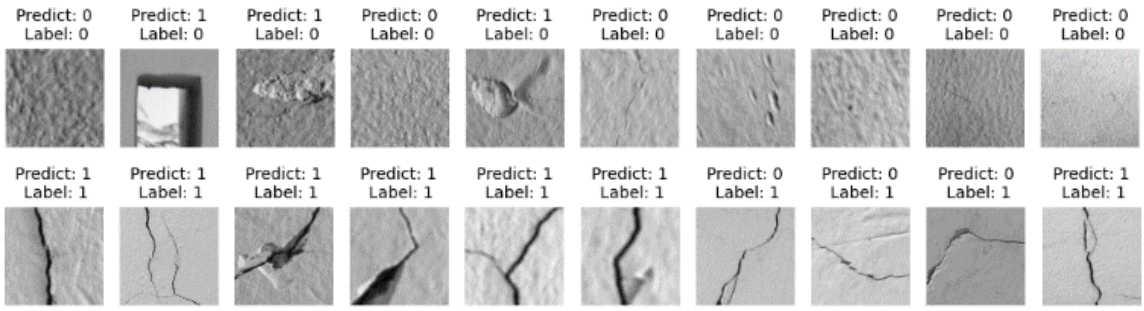
Positive

Negative

- Mô hình được huấn luyện với learning_rate = 0.1 và epoch = 700 cho ra Accuracy = 70%.



- Đồng thời mô hình được huấn luyện với learning_rate = 0.15 và epoch = 800 cũng cho ra được Accuracy = 70%.



VIII. Đánh giá

8.1 Ưu điểm

- Mô hình không quá phức tạp

- Thời gian huấn luyện ngắn (khoảng 30 phút, trên tập dữ liệu train gồm 24000 ảnh).

- Mô hình dự đoán tốt các ảnh có nhãn Positive.

- Accuracy trên tập validation và tập test cao (trên 90%).

8.2 Nhược điểm

- Mô hình dự đoán nhầm từ Negative thành Positive các trường hợp sau đây: khe hở giữa 2 bức tường, khoảng trống giữa hai bức tường, một bức tường nhưng được sơn 2 màu khác nhau, góc vuông giữa 2 bề mặt tường, tường bị tróc sơn.

Tài liệu tham khảo:

[1] Post 6 - Tiền xử lý ảnh bằng OpenCV – Nguyen Quoc Khanh - 阮 国 庆 (visionblog.github.io)

[2] Thị giác máy tính với OpenCV-Python Bài 4, Phần 3: Tạo ngưỡng hình ảnh (imc.org.vn)

[3] Machine Learning cơ bản (machinelearningcoban.com)

[4] 11.1. Feature Engineering — Deep AI KhanhBlog (phamdinhkhanh.github.io)

[5] Bài 6: Logistic Regression (Hồi quy Logistic) - Trí tuệ nhân tạo (trituenhantao.io)