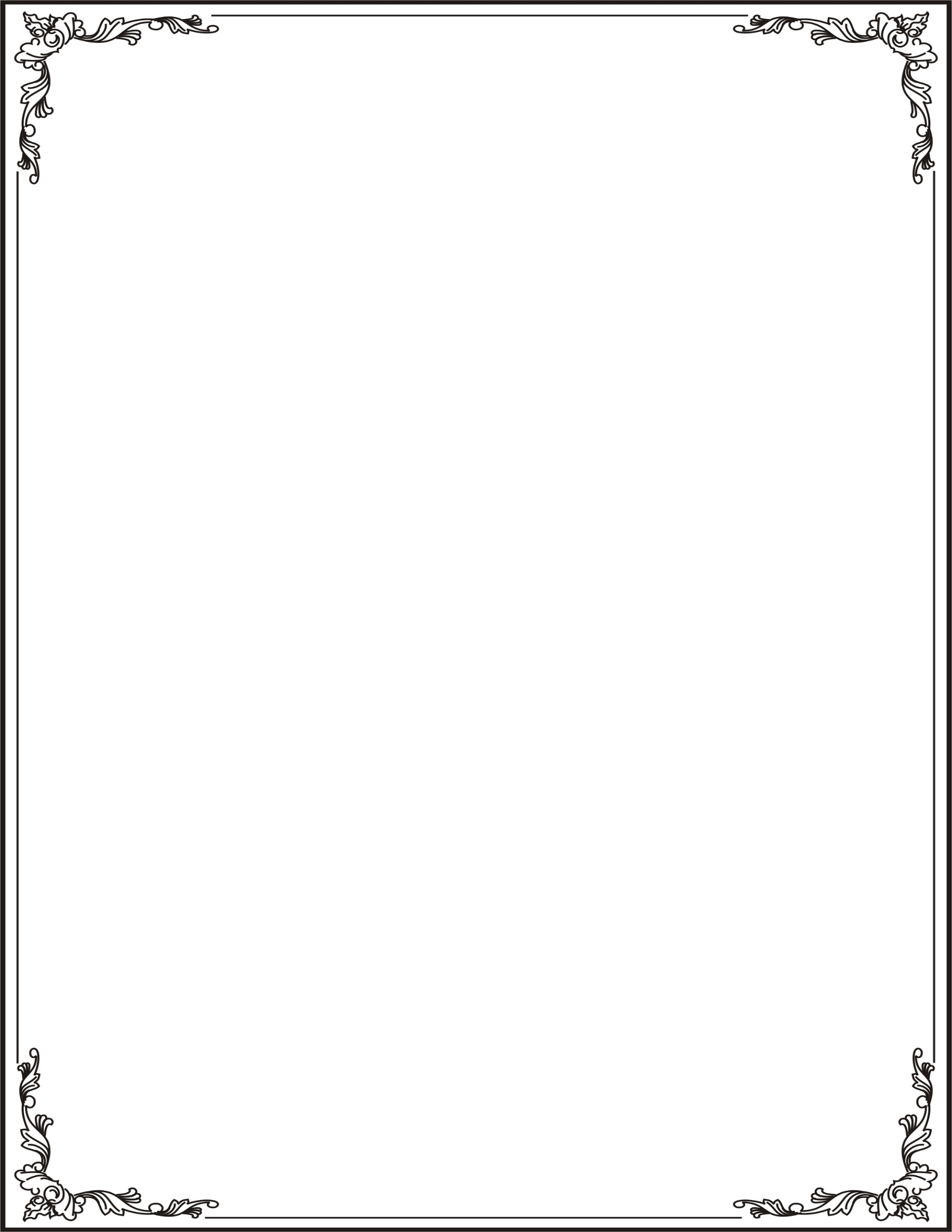
****

**TRƯỜNG ĐẠI HỌC LAO ĐỘNG - XÃ HỘI**

**KHOA GIÁO DỤC ĐẠI CƯƠNG-CHUYÊN NGÀNH CNTT**

**----------o0o----------**

****

**BÁO CÁO BÀI TẬP LỚN**

**MÔN : KHAI PHÁ DỮ LIỆU**

**Đề tài: Ứng dụng mạng Nơron trong nhận dạng hình ảnh.**

**Họ và tên sinh viên thực hiện** : Nguyễn Văn Hoàng

Bùi Ngọc Quang

**Mã sinh viên** : 1117091686

1117091607

**Giảng viên hướng dẫn** : Nguyễn Thị Vàn

**Hà Nội, tháng 5 năm 2024**

MỤC LỤC

[Mạng neural network AlexNet 3](#_Toc166588466)

[1. Tổng quan 3](#_Toc166588467)

[2. Kiến trúc 4](#_Toc166588468)

[2.1 Kiến trúc mạng 4](#_Toc166588469)

[2.2 Overlapping Max Pooling 7](#_Toc166588470)

[2.3 Local Response Normalization 7](#_Toc166588471)

[2.4 Relu Nonlinearity 8](#_Toc166588472)

[2.5 Sử dụng nhiều GPU 10](#_Toc166588473)

[3. Reducing Overfitting 10](#_Toc166588474)

[3.1 Data Augmentation 10](#_Toc166588475)

[3.2 Softmax 11](#_Toc166588476)

[3.3 Dropout 11](#_Toc166588477)

[Kết quả 12](#_Toc166588478)

[Kết luận 13](#_Toc166588479)

[Demo Ứng Dụng 14](#_Toc166588480)

[1. ` 14](#_Toc166588481)

[1.2. Mã huấn luyện mô hình 16](#_Toc166588482)

[2. Huấn luyện mô hình 18](#_Toc166588483)

[3.Chạy chương trình 19](#_Toc166588484)

[Tài lệu tham khảo 21](#_Toc166588485)

# Mạng neural network AlexNet

## Tổng quan

Mạng huấn luyện AlexNet là công trình đầu tiên phổ biến mạng CNN trong lĩnh vực Thị giác máy tính, cũng là một trong những mạng huấn luyện CNN nổi tiếng nhất nhờ thành tích ấn tượng mà nó đạt được trong cuộc thi nhận dạng ảnh quy mô lớn tổ chức vào năm 2012. Cuộc thi này có tên chính thức là ILSVRC – ImageNet Large Scale Visual Recognition Challenge, được ImageNet - một hãng CSDL ảnh - tổ chức thường niên và được coi là cuộc thi Olympics quy mô thế giới trong lĩnh vực Thị giác máy tính. Mục đích của cuộc thi là nhằm thử nghiệm các công nghệ mới giúp cho máy tính có thể hiểu, phân tích, phát hiện và nhận dạng các vật thể trong một bức ảnh.

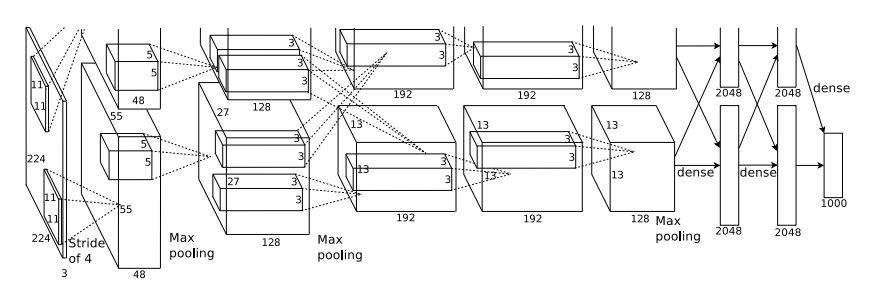
Nhiệm vụ chính của cuộc thi năm 2012 đặt ra mà các đội tham gia phải giải quyết là bài toán nhận dạng, với bộ dữ liệu huấn luyện lên đến 1,2 triệu ảnh được gán nhãn cho 1.000 hạng mục khác nhau. Nhóm SuperVision, gồm các thành viên Alex Krizhevsky, Ilya Sutskever và Geoff Hinton, cùng với mạng AlexNet của họ đã đạt được kết quả đáng kinh ngạc là chiến thắng áp đảo nhóm đứng thứ hai với độ chính xác chênh lệch đến hơn 10% (15,31% và 26,17%). Điều đặc biệt là mạng huấn luyện này chỉ nhận dữ liệu đầu vào là các giá trị điểm ảnh thô và không hề áp dụng bất kỳ phương pháp trích chọn đặc trưng nào, trong khi mọi hệ thống nhận dạng thị giác truyền thống đều phải gồm nhiều giai đoạn trích chọn đặc trưng hết sức tỉ mỉ, cẩn thận, thậm chí phải áp dụng nhiều mẹo để cải thiện chất lượng nhận dạng. Thiết kế kiến trúc mạng huấn luyện gần như một hộp đen, cộng với khả năng tự học các đặc trưng thông qua các lớp ẩn, đã khiến CNN nói riêng và Học sâu nói chung trở thành giải pháp mạnh mẽ nhất cho bài toán nhận dạng và phân loại vật thể cho tới bây giờ.

Từ năm 2012, mạng CNN trở thành cái tên gắn liền với cuộc thi và đã có rất nhiều mạng CNN nổi bật khác xuất hiện trong những năm sau đó VGG, GoogleNet hay Microsoft ResNet… Các mạng CNN càng ngày càng đạt độ chính xác cao hơn, tuy nhiên chúng có độ phức tạp và độ sâu lớn hơn rất nhiều, ví dụ mạng CNN có thể coi là tốt nhất hiện nay – ResNet – đã sử dụng đến 152 lớp tính toán. Sự phức tạp này yêu cầu khả năng tính toán lớn, thời gian huấn luyện lâu, và gây nhiều khó khăn trong việc cài đặt triển khai hệ thống, do đó mạng AlexNet đã được chọn làm cơ sở phát triển phiên bản 26 thử nghiệm ban đầu và việc cài đặt các mạng huấn luyện khác nhằm nâng cao chất lượng nhận dạng của hệ thống sẽ được thử nghiệm trong tương lai.

## 2. Kiến trúc

### 2.1 Kiến trúc mạng

Nhóm của Alex Krizhevsky đã công bố một bài báo với tiêu đề “ImageNet Classification with Deep Convolutional Networks” [20], đưa ra mô tả cụ thể về kiến trúc của mạng AlexNet cũng như cách thức cài đặt và sử dụng các lớp trong mạng để huấn luyện mô hình với bộ dữ liệu ảnh của ImageNet. Mạng có cấu trúc tương đối đơn giản nếu so với các mạng CNN hiện đại gần đây, bao gồm 5 lớp Tích chập và 3 lớp kết nối đầy đủ với các lớp giữa là các lớp lấy mẫu và ReLU, được huấn luyện song song trên hai card đồ họa GPU.



*Hình 1. Kiến trúc mạng AlexNet*

Mạng AlexNet bao gồm 8 lớp , bao gồm:

***Lớp thứ nhất (Tích chập):***

Đầu vào: Ảnh với kích thước 224 x 224 x 3 (3 là con số tương ứng với 3 màu đỏ, xanh lục, xanh lam trong hệ màu RGB thông thường)

Số bộ lọc: 96

Kích thước bộ lọc: 11 x 11 x 3

Bước trượt (Stride): 4

Đầu ra: (224/4) x (224/4) x 96 = 55 x 55 x 96, chia đều cho hai GPU

Lớp chuyển tiếp sang lớp 2 (Lấy mẫu tối đa): Đầu vào: 55 x 55 x 96, đầu ra: (55/2) x (55/2) x 96 = 27 x 27 x 96

***Lớp thứ hai (Tích chập):***

Đầu vào: 27 x 27 x 96

Số bộ lọc: 256

Kích thước bộ lọc: 5 x 5 x 48

Đầu ra: 27 x 27 x 256, chia đều cho hai GPU

***Lớp thứ ba, bốn,năm (Tích chập):***

Tương tự như với lớp 1 và lớp 2 với các kích thước bộ lọc lần lượt là 3 x 3 x 256, 3 x 3 x 384 và 3 x 3 x 384. Toàn bộ các lớp tính toán này đều được chia đều cho hai GPU để tăng tốc độ xử lý. Đầu ra cuối cùng qua lớp Tích chập thứ 5 là dữ liệu với kích thước 13 x 13 x 128, dữ liệu này sau khi đi qua một lớp Lẫy mẫu tối đa cuối cùng sẽ được dùng làm đầu vào cho các lớp sau đó là các lớp Kết nối đầy đủ

***Lớp thứ sáu (lớp kết nối đầy đủ):***

Đầu vào: 6 x 6 x 256

Số nơ-ron: 4096

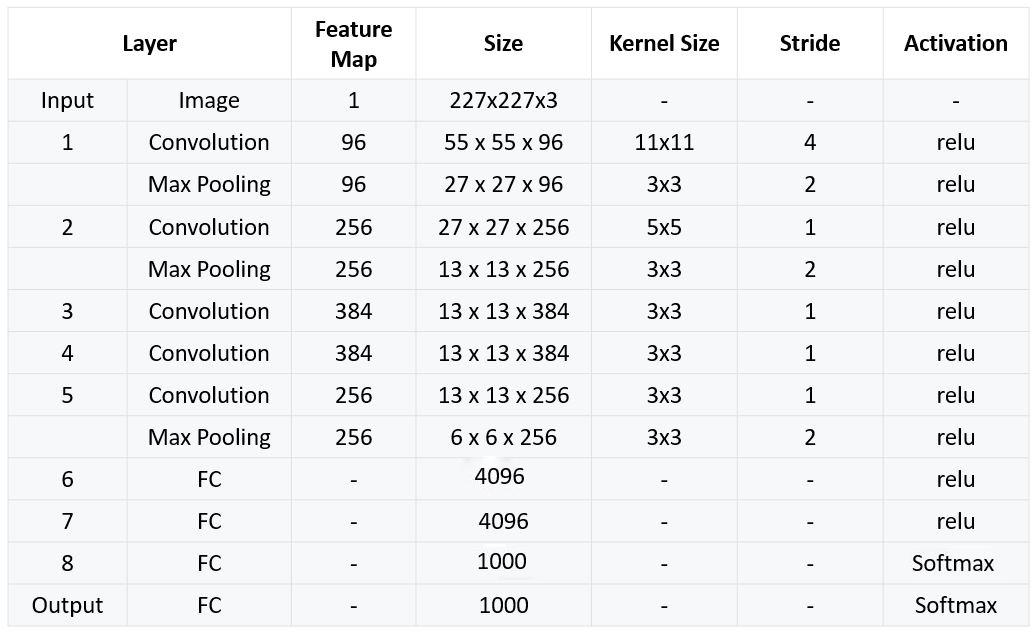
***Lớp thứ bảy (lớp kết nối đầy đủ):***

Đầu vào: 6 x 6 x 256

Số nơ-ron: 4096

***Lớp thứ tám (lớp kết nối đầy đủ):***

Lớp cuối cùng trong mạng AlexNet này có 1000 nơron, tương ứng với 1000 lớp khác nhau mà bộ huấn luyện cần nhận dạng.



Những convolution layer ( hay còn gọi với tên khác là các filter) rút trích các thông tin hữu ích trong các bức ảnh. Trong một convolution layer bất kỳ thường bao gồm nhiều kernel có cùng kích thước. Ví dụ như convolution layer đầu tiên của AlexNet chứa 96 kernel có kích thước 11x11x3. Thông thường thì width và height của một kernel bằng nhau, và độ sâu (depth) thường bằng số lượng kênh màu.

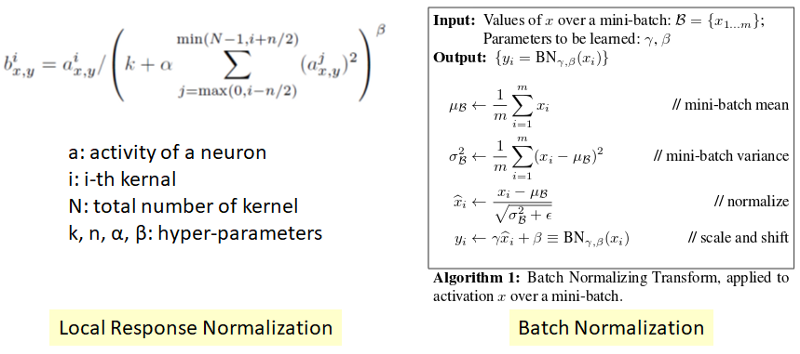
Convolutional 1 và convolution 2 kết nối với nhau qua một Overlapping Max Pooling ở giữa. Tương tự như vậy giữa convolution 2 và convolution 3. Convolutional 3, convolution 4, convolution 5 kết nối trực tiếp với nhau, không thông qua trung gian. Convolutional 5 kết nối fully connected layter 1 thông qua một Overlapping Max pooling, tiếp theo mà một fully connected layter nữa. Và cuối cùng là một bộ phân lớp softmax với 1000 lớp nhãn (các bạn có thể xem hình kiến trúc mạng AlexNet ở trên để có cái nhìn tổng quát hơn).

ReLU nonlinerity được sử dụng sau tất các các convolution và fully connected layer. Trước đây, ReLU nonlinerity của lớp convolution 1 và 2 thường theo sau bởi một bước chuẩn hoá cục bộ (local normalization) rồi mới thực hiện pooling. Tuy nhiên, các nghiên cứu sau đó nhận thấy rằng việc sử dụng normalization không thật sự hữu ích. Do vậy chúng ta sẽ không đi chi tiết về vấn đề đó.

### 2.2 Overlapping Max Pooling

Max Pooling layer thường được sử dụng để giảm chiều rộng và chiều dài của một tensor nhưng vẫn giữ nguyên chiều sâu. Overlapping Max Pool layer cũng tương tự như Max Pool layer, ngoại trừ việc là một window của bước này sẽ có một phần chồng lên window của bước tiếp theo. Tác giả sử dụng pooling có kích thước 3x3 và bước nhảy là 2 giữa các pooling. Nghĩa là giữa pooling này và pooling khác sẽ overlapping với nhau 1 pixel. Các thí nghiệm thực tế đã chứng minh rằng việc sử dụng overlapping giữa các pooling giúp giảm độ lỗi top-1 error 0.4% và top-5 error là 0.3% khi so với việc sử dụng pooling có kích thước 2x2 và bước nhảy 2 (vector output của cả hai đều có số chiều bằng nhau).

### 2.3 Local Response Normalization

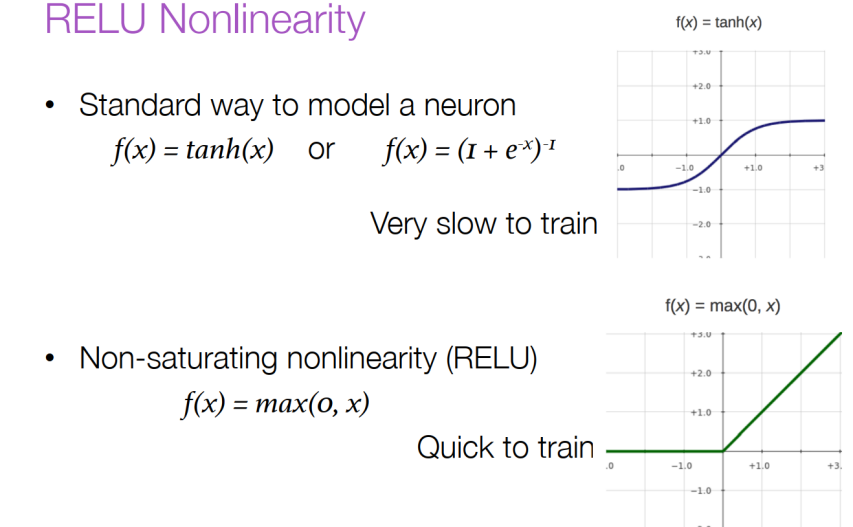


Trong mạng AlexNet, nhóm tác giả sử dụng hàm chuẩn hóa là Local Response Normalization. Hàm này không phải là Batch Normalization mà các bạn hay sử dụng ở thời điểm hiện tại (xem hình ở trên, hai hàm có công thức tính toán hoàn toàn khác nhau). Việc sử dụng chuẩn hóa (Normalization) giúp tăng tốc độ hội tụ. Ngày nay, chúng ta không còn sử dụng Local Response Normalization nữa. Thay vào đó, chúng ta sử dụng Batch Normalization làm hàm chuẩn hóa.

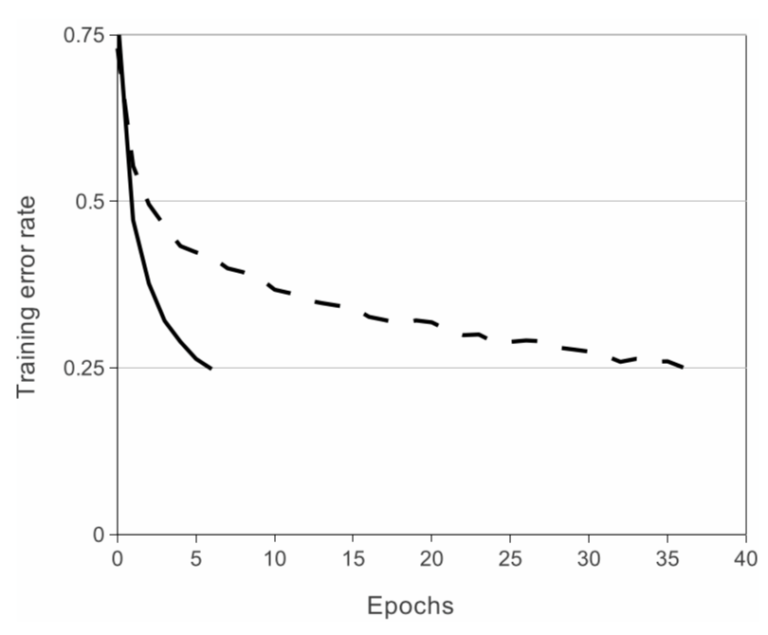
Với việc sử dụng hàm chuẩn hóa Local Response Normalization, độ lỗi top-1 error rate giảm 1.4%, top-5 giảm 1.2%.

### 2.4 Relu Nonlinearity

Một cải tiến quan trọng khác của AlexNet là việc sử dụng hàm phi tuyến ReLU. Trước đây, các nhóm nghiên cứu khác thường sử dụng hàm kích hoạt là hàm Tanh hoặc hàm Sigmoid để huấn luyên mô hình neural network. AlexNet chỉ ra rằng, khi sử dụng ReLU, mô hình deep CNN sẽ huấn luyện nhanh hơn so với viêc sử dụng tanh hoặc sigmoid.Sau đây sẽ là một số hình ảnh so sánh về hàm ReLU và tanh.

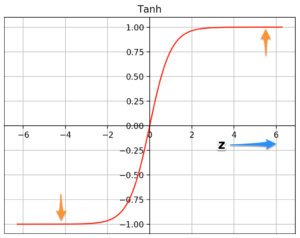


Hình bên dưới được rút ra từ bài báo chỉ ra rằng với việc sử dụng ReLU (đường nét liền trong hình), AlexNet đạt độ lỗi 25% trên tập huấn luyện và nhanh hơn gấp 6 lần so với mô hình tương tự nhưng sử dụng Tanh (đường nét đứt trong hình). Thí nghiệm trên sử dụng tập dữ liệu CIFAR-10 để huấn luyện.



Để hiểu rõ hơn lý do vì sao ReLU lại nhanh hơn so với các hàm khác, chúng ta hãy đối sánh hình dạng giá trị output của các hàm trên.

Công thức của ReLU là: f(X) = max(0,x)



Nhìn kỹ vào hình trên, ta có nhận xét rằng: hàm tanh đạt giá trị bão hoà khi giá trị z >2.5 và z < -2.5 (số 2.5 là số cảm tính của mình). Và tại vùng |z|>2.5, thì độ dốc của hàm hầu như gần như bằng 0, |z| càng lớn thì độ dốc càng gần 0 hơn. Vì lý do này nên gradient descent sẽ hội tụ chậm. Còn đối với hàm ReLU, với giá trị z dương thì độ dốc của hàm không gần bằng 0 như hàm tanh. Điều này giúp cho việc hội tụ xảy ra nhanh hơn. Với giá trị z âm, độ dốc bằng 0, tuy nhiên, hầu hết các giá trị của các neural trong mạng thường có giá trị dương, nên trường hợp âm ít (hiếm) khi xảy ra. ReLU huấn luyện nhanh hơn so với sigmoid cũng bởi lý do tương tự.

### 2.5 Sử dụng nhiều GPU

Tại năm 2012, nhóm tác giả sử dụng card đồ họa NIVIDIA GTX 580 có 3GB bộ nhớ RAM. Cho nên, để có thể huấn luyện được mô hình AlexNet trên GPU, mô hình cần sử dụng 2 GPU.Vì vậy việc sử dụng 2 hoặc nhiều GPU là do vấn đề thiếu bộ nhớ, chứ không phải là vấn đề tăng tốc quá trình train hơn so với 1 GPU

Ngoài ra, do giới hạn của GPU, nên mô hình AlexNet được tách ra làm 2 phần, mỗi phần được huấn luyện trên 1 GPU. Phiên bản 1 GPU của mô hình có tên là CaffeNet, và đòi hỏi chúng ta phải sử dụng GPU có bộ nhớ RAM lớn hơn hoặc bằng 6GB.

## Reducing Overfitting

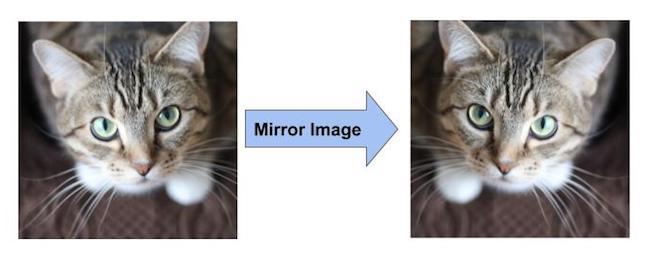
Vậy overfitting là gì?

Nói một cách dễ hiểu , overfitting giống như tình trạng học vẹt đối với con người. Có nghĩa là neuron network sẽ hoạt động tốt trên tập huấn luyện nhưng không rút ra được vấn đề . Trong mạng Alexnet , ta sẽ dung nhiều cách khác nhau để giảm overfitting.

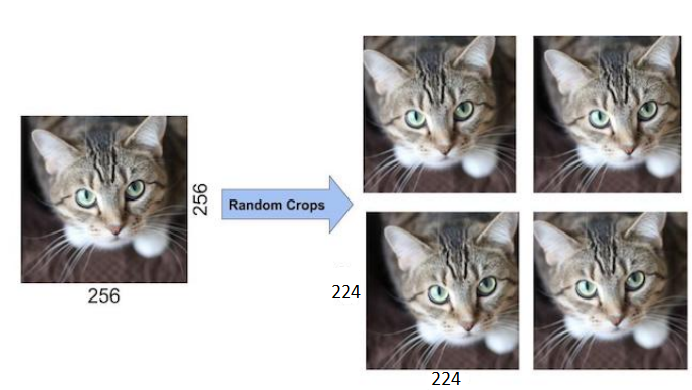
### 3.1 Data Augmentation

Việc sử dụng nhiều biến thể khác nhau của một bức hình có thể giúp ngăn mô hình không bị overfitting. Với việc sử dụng nhiều biến thể của 1 bức hình, bạn bắt ép mô hình không học vẹt dữ liệu. Có nhiều cách khác nhau để sinh ra dữ liệu mới dựa vào dữ liệu có sẵn. Một vài các mà nhóm AlexNet đã sử dụng là:

Data Augmentation by Mirroring: một số biến thể được sử dụng đó là lấy ảnh gương , cho dễ hiểu là xoay ảnh qua trục y.



Một cách khác được sử dụng là crop ảnh . Crop từ bức ảnh ban đầu kích thước 256x256 ban đầu xuống 224x224. Ta có thể tang số lượng dữ liệu , nhìn bức ảnh dưới nhiều góc độ khác nhau.



Viêc sử dụng Data Augmentation , chúng ta đang dạy cho mô hình rằng nhìn con mèo dưới nhiều góc độ qua gương , hay góc trái , góc phải nó vẫn là con mèo.

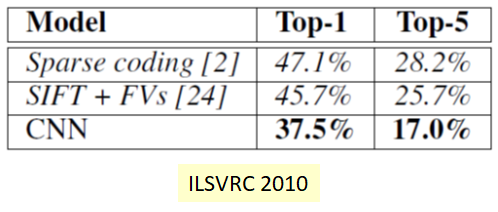
### 3.2 Softmax

Việc sử dụng hàm softmax hay giảm sắc độ của các màu RGB cũng sẽ làm giảm phần nào overfitting .

### 3.3 Dropout

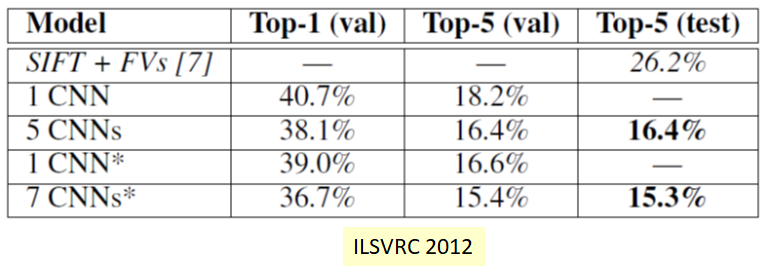
Việc tập huấn luyện chứa gần 60 triệu tham số , việc overfitting lớn là điều dễ hiểu . Các tác giả của AlexNet còn sử dụng 1 kỹ thuật gọi là Dropout - một kỹ thuật khá đơn giản . Một neural sẽ có xác suất bị loại khỏi mô hình là 0.5 , nó sẽ không được tham gia vào quá trình lan truyền tiến và lan truyền ngược . Cho nên, mỗi giá trị input sẽ đi qua một kiến trúc mạng khác nhau , điều này sẽ khiến giá trị output bị scaled , ta sẽ tang gấp đôi lần lập cần thiết để đạt được độ hội tụ , nhưng đổi lại overfitting sẽ được giảm đi .

### Kết quả



*Độ lỗi của AlexNet trên ILSVRC 2010*

Trong cuộc thi ILSVRC 2010, AlexNet đạt độ chính xác top-1 error 37.5% và top-5 error là 17.0%, kết quả này tốt hơn vượt trội so với các cách tiếp cận khác.



*Độ lỗi của AlexNet trên ILSVRC 2012*

Đến cuộc thi ILSVRC 2012, độ lỗi của AlexNet trên tập validation giảm còn 18.2%.

Nếu lấy trung bình của dự đoán trên 5 mạng AlexNet được huấn luyện khác nhau, độ lỗi giảm còn 16.4%. Các lấy trung bình trên nhiều hơn 1 mạng CNN là một kỹ thuật boosting và được sử dụng trước đó ở bài toán phân loại số của mạng LeNet.

Ở dòng số 3 là mạng AlexNet nhưng được thêm 1 convolution layer nữa (nên được ký hiệu là 1CNN\*), độ lỗi trên tập validation giảm còn 16.4%.

Nếu lấy kết quả trung bình của 2 mạng neural net được chỉnh sửa (thêm 1 convolution layer) và 5 mạng AlexNet gốc (=> chúng ta có 7CNN\*), độ lỗi trên tập validation giảm xuống 15.4%

### Kết luận

Chúng ta đã tìm hiểu qua về mạng Alexnet một mạng có cấu trúc lớn và phức tạp , và những đóng góp chính của nó trong CNN.

Qua bài nghiên cứu , ta sẽ rút ra những kiến thức , đóng góp cho việc học tập và nghiên cứu , phát triển ứng dụng của AI trong đời sống , phát triển những mạng lớn hơn.

# Demo Ứng Dụng

1. `

import cv2

import numpy as np

from keras.preprocessing import image

from keras.models import load\_model

import os

# Khởi tạo kết nối với camera

vid = cv2.VideoCapture(0)

print("Camera connection successfully established")

# Khởi tạo biến đếm số lượng ảnh đã chụp

i = 0

# Danh sách các lớp mà mô hình sẽ dự đoán

classes = ['Chuối','Dâu tây','Dứa','Khế','Xoài']

# Tải mô hình máy học đã được đào tạo từ tệp 'my\_model.keras'

new\_model = load\_model('my\_model.keras')

# Bắt đầu vòng lặp vô hạn để liên tục chụp và xử lý các khung hình từ camera

while(True):

    # Đọc một khung hình từ camera

    r, frame = vid.read()

    # Hiển thị khung hình lên màn hình

    cv2.imshow('frame', frame)

    # Lưu khung hình hiện tại vào một tệp ảnh với tên tăng dần

    cv2.imwrite(r'D:\KPDL\_T2\Do\_anst2\csudai\\final' + str(i) + ".jpg", frame)

    # Tải ảnh từ tệp vừa lưu và thay đổi kích thước của ảnh thành 64x64 pixels

    test\_image = image.load\_img(r'D:\KPDL\_T2\Do\_anst2\csudai\\final' + str(i) + ".jpg", target\_size=(64, 64))

    # Chuyển ảnh thành mảng numpy

    test\_image = image.img\_to\_array(test\_image)

    # Mở rộng số chiều của mảng numpy để phù hợp với đầu vào của mô hình

    test\_image = np.expand\_dims(test\_image, axis=0)

    # Dự đoán lớp của ảnh

    result = new\_model.predict(test\_image)

    # Lấy kết quả dự đoán đầu tiên

    result1 = result[0]

    # Duyệt qua các lớp trong danh sách 'classes' để tìm lớp có xác suất dự đoán cao nhất

    for y in range(5):

        if result1[y] == 1.:

            break

    # Lấy tên của lớp được dự đoán từ danh sách 'classes'

    prediction = classes[y]

    # Hiển thị tên của lớp được dự đoán

    print(prediction)

    # Xóa tệp ảnh vừa lưu sau khi đã sử dụng

    os.remove(r'D:\KPDL\_T2\Do\_anst2\csudai\\final' + str(i) + ".jpg")

    # Tăng biến đếm để đảm bảo tên của các tệp ảnh mới không trùng lặp

    i = i + 1

    # Đợi một phím được nhấn, nếu phím là 'q' thì thoát khỏi vòng lặp

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

# Giải phóng tài nguyên camera

vid.release()

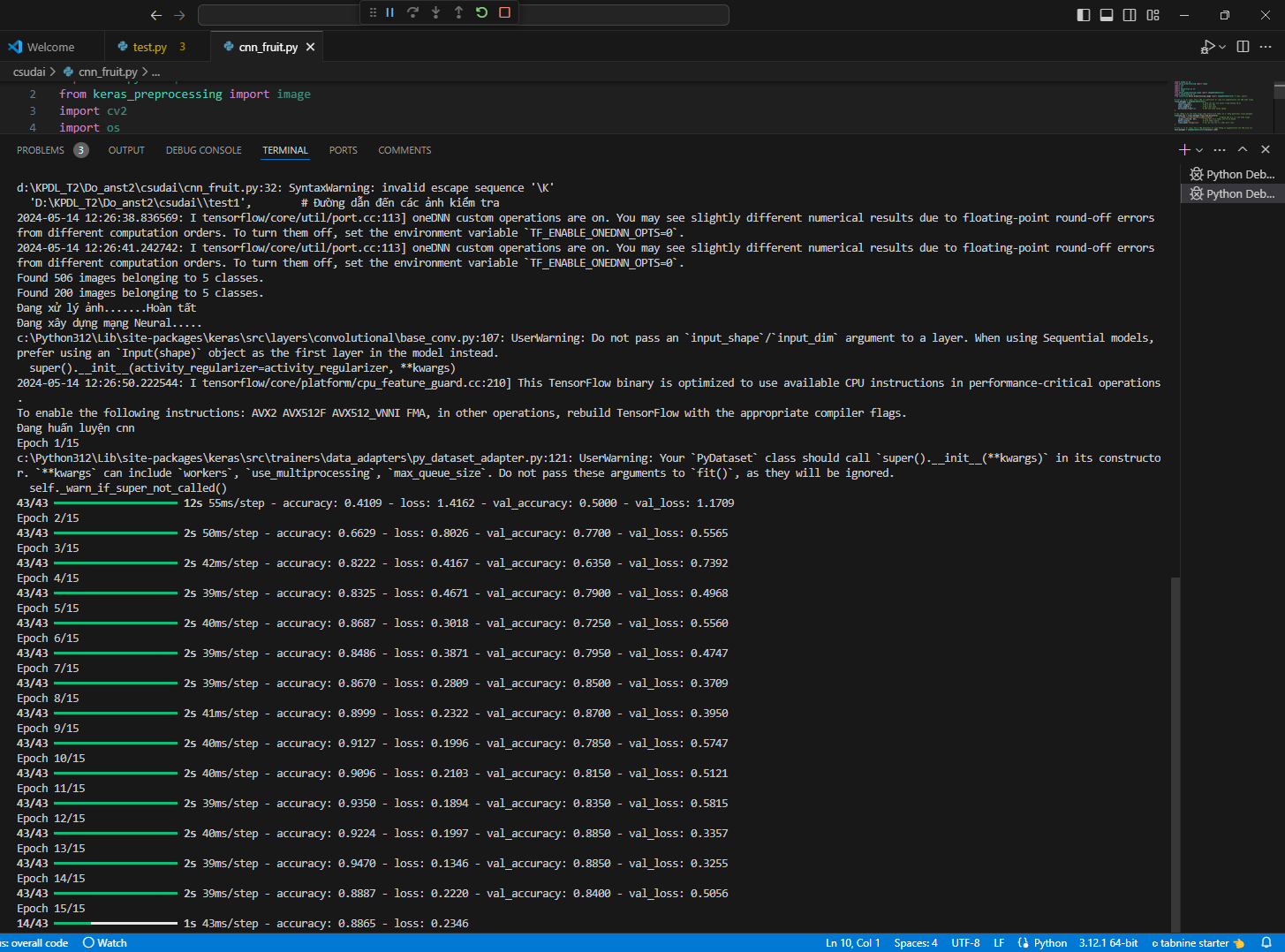
# Đóng tất cả các cửa sổ hiển thị hình ảnh của OpenCV

cv2.destroyAllWindows()

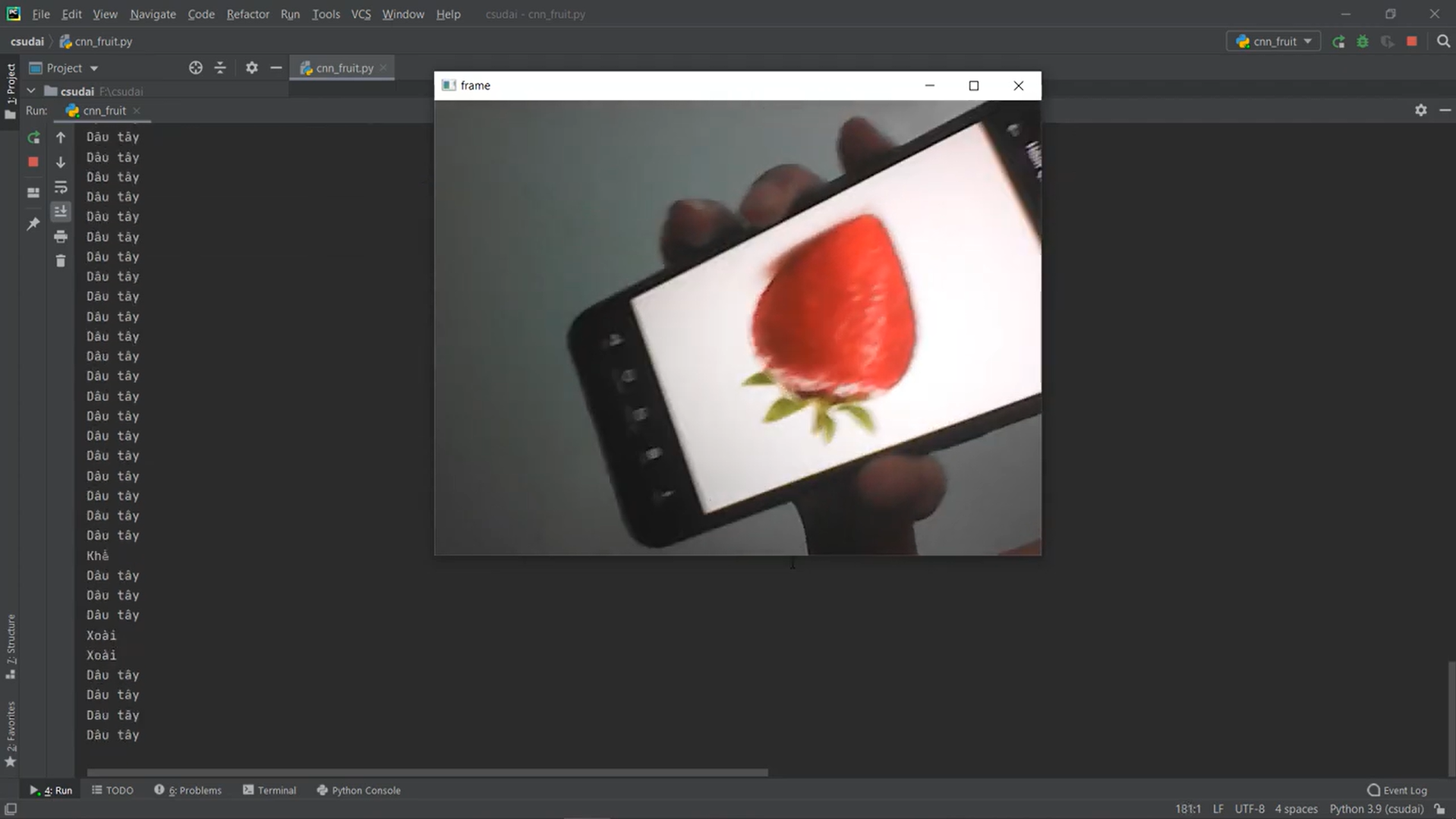
* 1. Mã huấn luyện mô hình

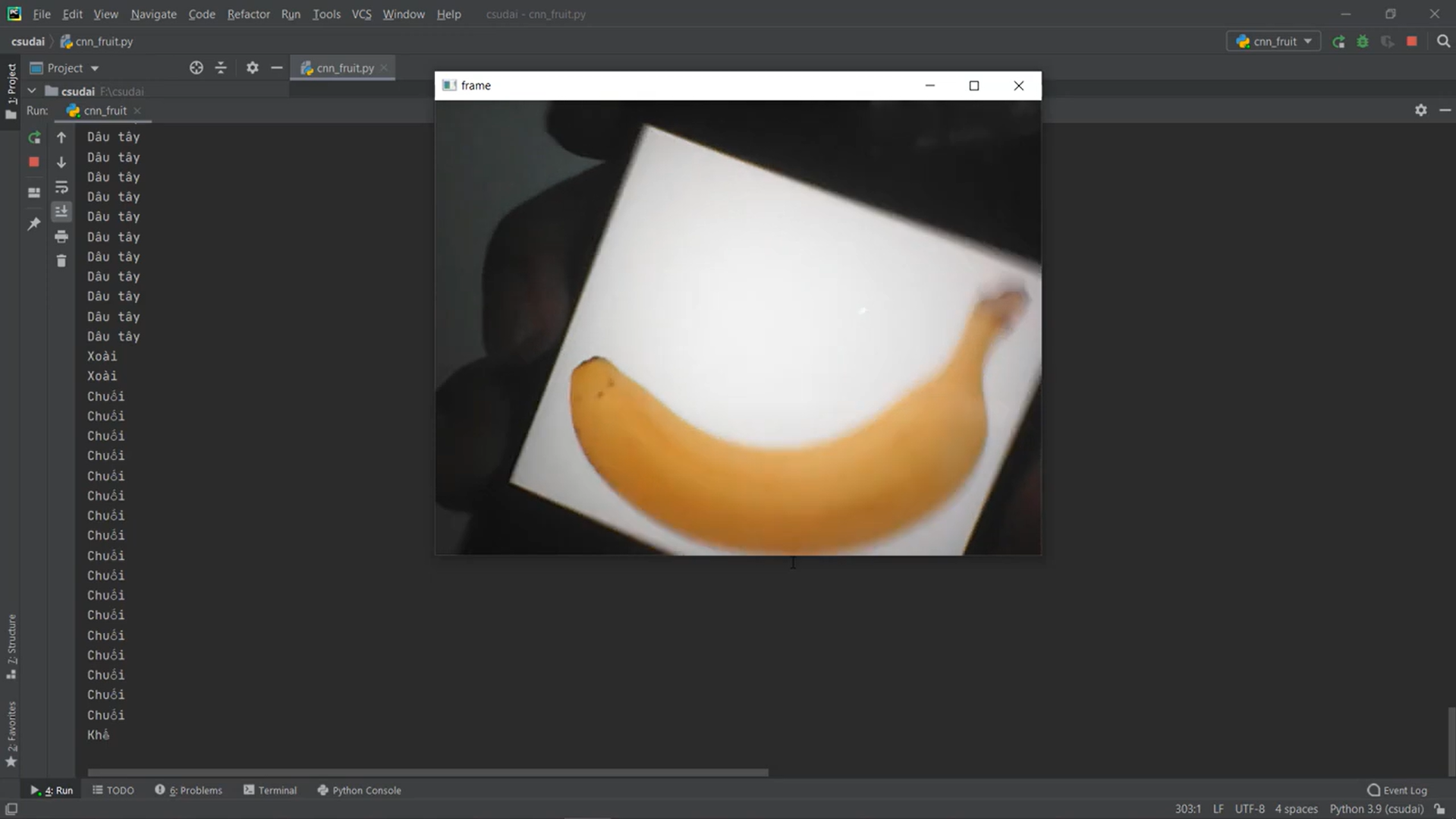
1. import numpy as np
2. from keras\_preprocessing import image
3. import cv2
4. import os
5. import tensorflow as tf
6. import time
7. from keras\_preprocessing.image import ImageDataGenerator
8. import tensorflow as tf
9. from tensorflow.keras.preprocessing.image import ImageDataGenerator # type: ignore
10. # Tiền xử lý dữ liệu: thiết lập các generator dữ liệu với augmentation cho tập huấn luyện
11. train\_datagen = ImageDataGenerator(
12. rescale=1./255,             # Chia lại giá trị pixel trong khoảng [0,1]
13. shear\_range=0.2,            # Biến đổi cắt
14. zoom\_range=0.2,             # Biến đổi zoom
15. horizontal\_flip=True        # Lật ảnh theo chiều ngang
16. )
17. # Lưu lượng hình ảnh huấn luyện theo batch kích thước 12 sử dụng generator train\_datagen
18. training\_set = train\_datagen.flow\_from\_directory(
19. 'D:\\KPDL\_T2\\Do\_anst2\\csudai\\train1',       # Đường dẫn đến các ảnh huấn luyện
20. target\_size=(64, 64),       # Thay đổi kích thước ảnh thành 64x64
21. batch\_size=12,              # Kích thước batch
22. class\_mode='categorical'    # Chế độ lớp cho các nhãn phân loại
23. )
24. # Tiền xử lý dữ liệu: thiết lập generator dữ liệu không có augmentation cho tập kiểm tra
25. test\_datagen = ImageDataGenerator(rescale=1./255)
26. # Lưu lượng hình ảnh kiểm tra theo batch kích thước 12 sử dụng generator test\_datagen
27. test\_set = test\_datagen.flow\_from\_directory(
28. 'D:\KPDL\_T2\Do\_anst2\csudai\\test1',        # Đường dẫn đến các ảnh kiểm tra
29. target\_size=(64, 64),       # Thay đổi kích thước ảnh thành 64x64
30. # target\_size=(256, 256),       # Thay đổi kích thước ảnh thành 64x64
31. batch\_size=12,              # Kích thước batch
32. # batch\_size=36,              # Kích thước batch
33. class\_mode='categorical'    # Chế độ lớp cho các nhãn phân loại
34. )
35. # Xác định các lớp cho việc phân loại
36. classes = ['Chuối', 'Dâu tây', 'Dứa', 'Khế', 'Xoài']
37. # In thông điệp trạng thái
38. print("Đang xử lý ảnh.......Hoàn tất")
39. # Xây dựng mô hình CNN
40. cnn = tf.keras.models.Sequential()
41. # In thông điệp trạng thái
42. print("Đang xây dựng mạng Neural.....")
43. # Thêm lớp tích chập đầu tiên với activation ReLU
44. cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu', input\_shape=[64, 64, 3]))
45. # Thêm lớp max pooling
46. cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))
47. # Thêm lớp tích chập thứ hai với activation ReLU
48. cnn.add(tf.keras.layers.Conv2D(filters=32, kernel\_size=3, activation='relu'))
49. # Thêm lớp max pooling
50. cnn.add(tf.keras.layers.MaxPool2D(pool\_size=2, strides=2))
51. # Làm phẳng các feature map
52. cnn.add(tf.keras.layers.Flatten())
53. # Thêm các lớp kết nối đầy đủ với activation ReLU
54. cnn.add(tf.keras.layers.Dense(units=32, activation='relu'))
55. cnn.add(tf.keras.layers.Dense(units=64, activation='relu'))
56. cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
57. cnn.add(tf.keras.layers.Dense(units=256, activation='relu'))
58. cnn.add(tf.keras.layers.Dense(units=256, activation='relu'))
59. # Thêm lớp đầu ra với activation softmax cho việc phân loại
60. cnn.add(tf.keras.layers.Dense(units=5, activation='softmax'))
61. # Biên dịch mô hình với optimizer Adam và loss categorical cross-entropy
62. cnn.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])
63. # In thông điệp trạng thái
64. print("Đang huấn luyện cnn")
65. # Huấn luyện mô hình với dữ liệu huấn luyện và kiểm tra
66. cnn.fit(x=training\_set, validation\_data=test\_set, epochs=15)
67. # Lưu mô hình đã huấn luyện
68. cnn.save("my\_model.keras")

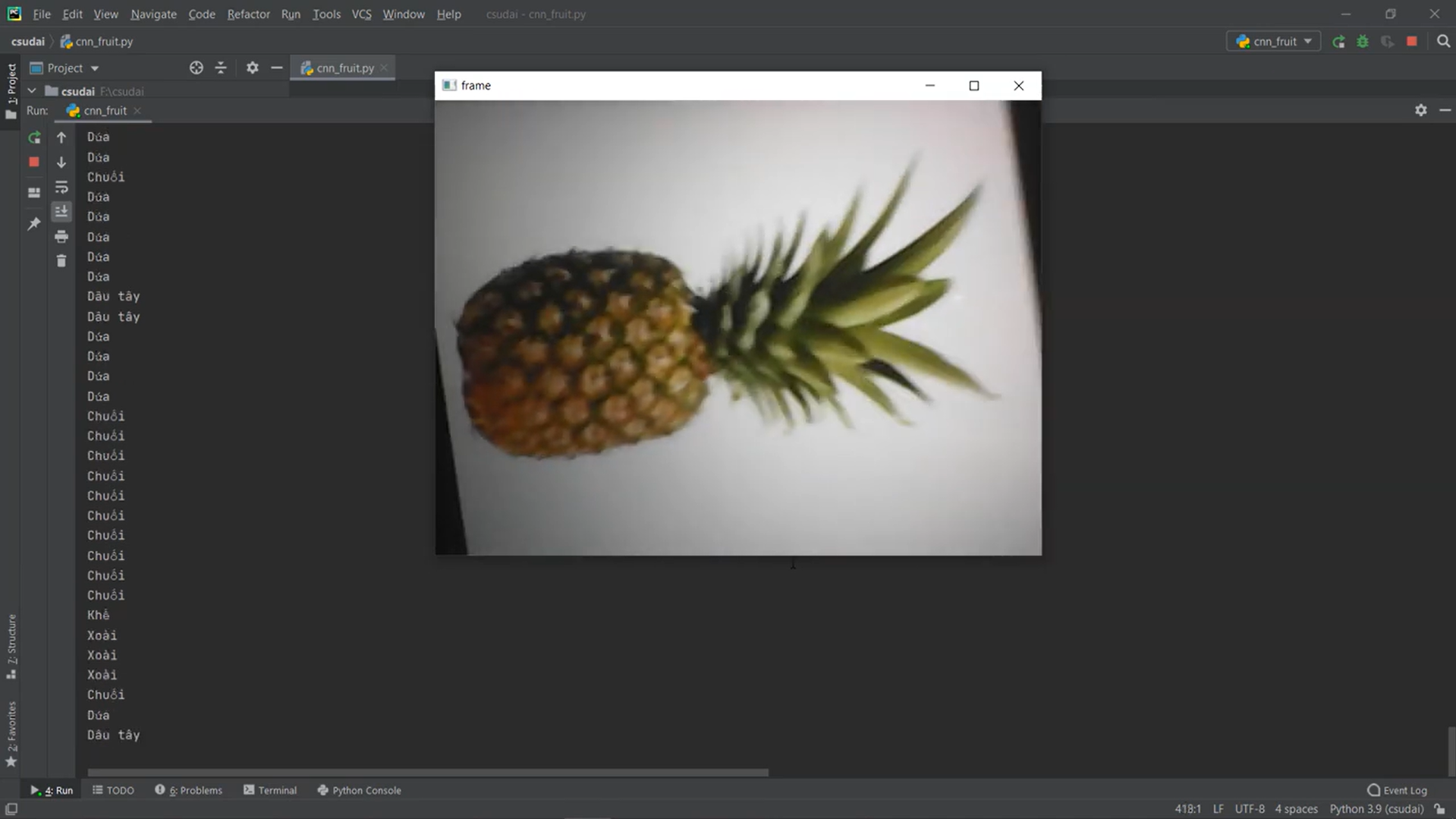
## 2. Huấn luyện mô hình

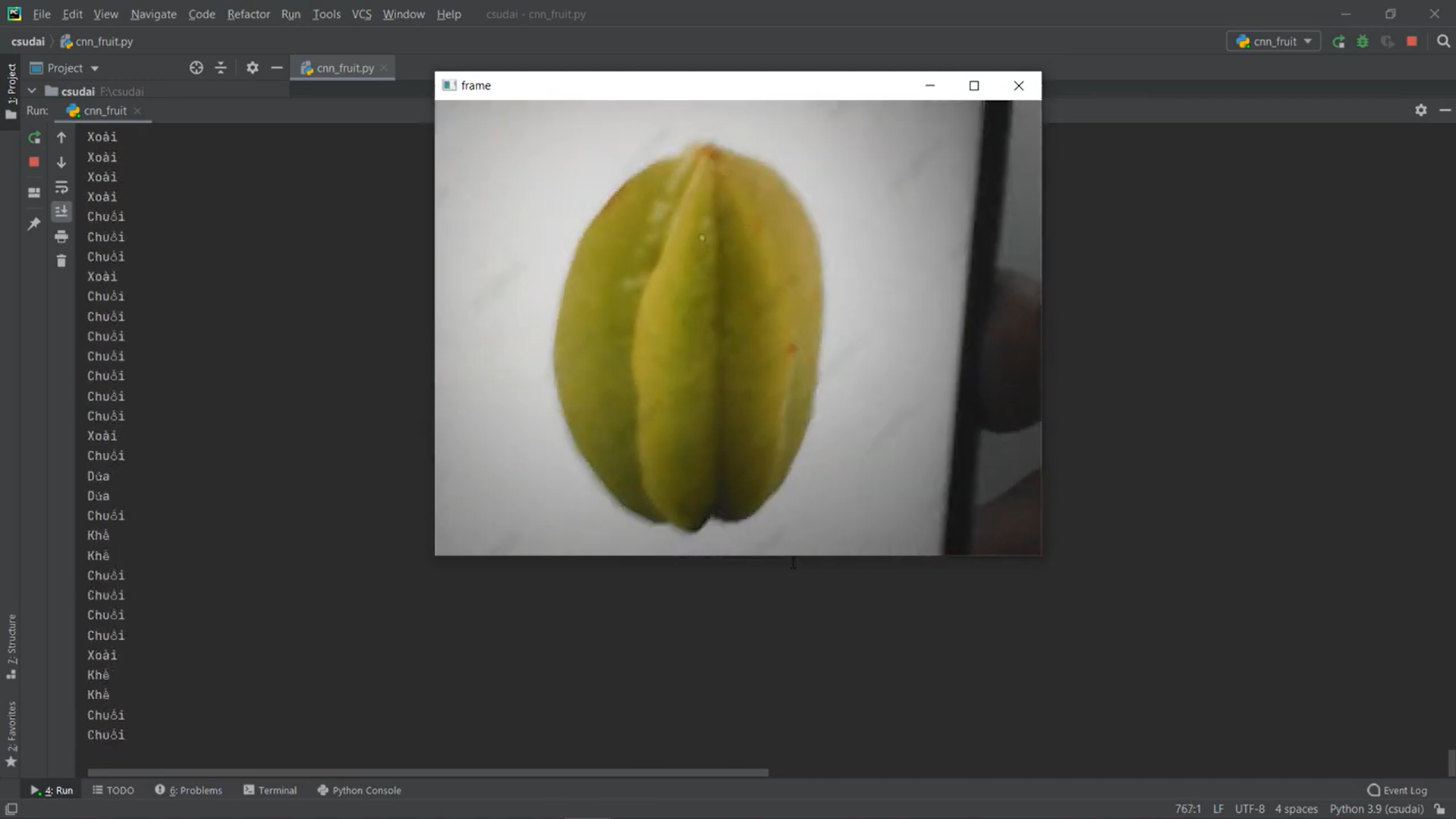


## 3.Chạy chương trình









# Tài lệu tham khảo

<https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

<http://www.image-net.org/challenges/LSVRC/>

ImageNet Classification with Deep Convolutional Neural Networks by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, 2012