

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**  
**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**  
**KHOA CƠ KHÍ CHẾ TẠO MÁY**



**HCMUTE**

**BÁO CÁO GIỮA KÌ**  
**ARTIFICIAL INTELLIGENCE**  
**NHẬN DIỆN HÌNH ẢNH BẰNG PHƯƠNG PHÁP DEEP LEARNING**  
**(Convolutional Neural Network - CNN)**

<b>GIẢNG VIÊN HƯỚNG DẪN:</b>	<b>PGS.TS NGUYỄN TRƯỜNG THỊNH</b>
<b>Mã Học Phần:</b>	<b>222ARIN337629E</b>
<b>HỌ &amp; TÊN SINH VIÊN:</b>	<b>Nguyễn Ngọc Nhân</b>
<b>MSSV:</b>	<b>20146262</b>

**TP Hồ Chí Minh, ngày 30 tháng 4 năm 2023**

# Mục lục

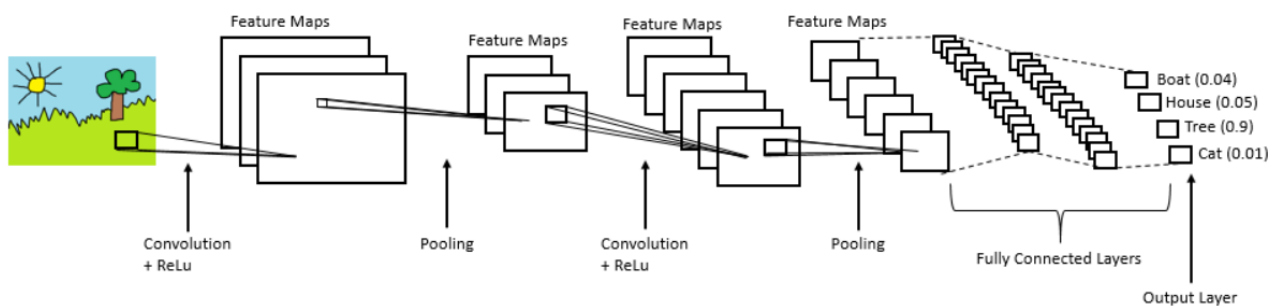
<b>1</b>	<b>Introduction - Giới thiệu</b>	<b>3</b>
<b>2</b>	<b>Methodology - Phương pháp</b>	<b>3</b>
2.1	Convolution Layer - Lớp tính chập . . . . .	3
2.2	Stride - Bước nhảy . . . . .	4
2.3	Padding - Đường viền . . . . .	5
2.4	Rectified Linear Unit (Relu) - Hàm phi tuyến . . . . .	5
2.5	Pooling Layer - Lớp gộp . . . . .	6
2.6	Gradient Descent . . . . .	6
2.7	Forward Pass and Backward Pass - chạy thuận và chạy nghịch . . . . .	7
2.8	Batch size, Iteration, Epoch . . . . .	7
<b>3</b>	<b>Implementation - Thực hiện</b>	<b>8</b>
3.1	Recognition of all members of class from face images (you collected) . . . . .	9
3.2	5 kinds of flowers (rose, lotus, water lily, apricot, daisy, pink) . . . . .	15
3.3	10 Vietnamese dishes (bún bò, chè, bánh xèo.....) . . . . .	20
3.4	VN banknotes (5000vnd, 10000vnd, 20k, 50k, 100k, 500k) . . . . .	25
3.5	Prediction of your future based on hand palm outline or face or fingerprint. . . . .	30
<b>4</b>	<b>Results - Kết quả</b>	<b>35</b>
<b>5</b>	<b>Conclusions - Kết luận</b>	<b>35</b>
<b>6</b>	<b>References</b>	<b>36</b>

# 1 Introduction - Giới thiệu

- Convolutional Neural Network (CNN – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning (DL) tiên tiến để nhận dạng và phân loại hình ảnh. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.

- CNN được áp dụng rộng rãi trong nhiều lĩnh vực hiện nay, bao gồm việc nhận diện khuôn mặt, phân loại bệnh qua ảnh chụp từ y khoa, phát hiện các object (vật thể) qua các video và hình ảnh. Với sự phát triển liên tục, CNN đã đóng vai trò quan trọng trong việc giải quyết các vấn đề về lĩnh vực thị giác máy tính.

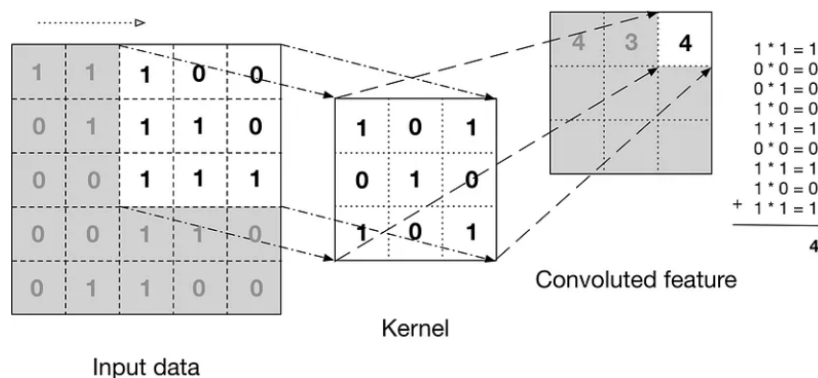
## 2 Methodology - Phương pháp







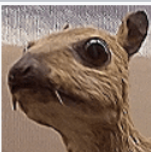
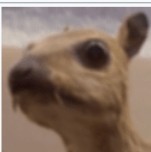
### 2.1 Convolution Layer - Lớp tích chập

- Là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Nó sử dụng phép tính toán convolution (tích chập) giữa các ma trận. Tức là, nó áp dụng một kernel (còn được gọi là filter hoặc weight) đến một vùng của ảnh đầu vào để tạo ra một feature map (bản đồ đặc trưng) mới.

- Việc tính toán này được thực hiện bằng cách áp dụng phép nhân ma trận giữa kernel và các phần tử tương ứng trên vùng ảnh đầu vào, sau đó tính tổng các kết quả và lưu vào feature map. Quá trình này được thực hiện trên toàn bộ ảnh đầu vào để tạo ra nhiều feature map khác nhau, từ đó trích xuất các đặc trưng của ảnh để phục vụ cho các mục đích như phân loại, nhận dạng, phát hiện.

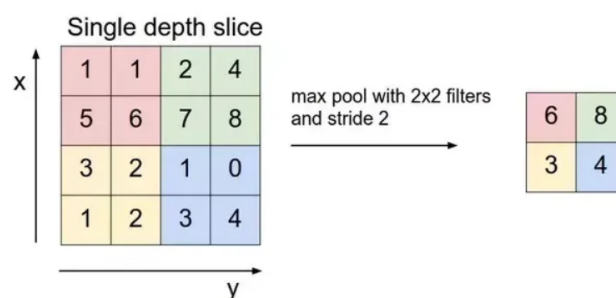


- Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể giúp chúng ta tìm ra cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc.
- Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.

Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

## 2.2 Stride - Bước nhảy

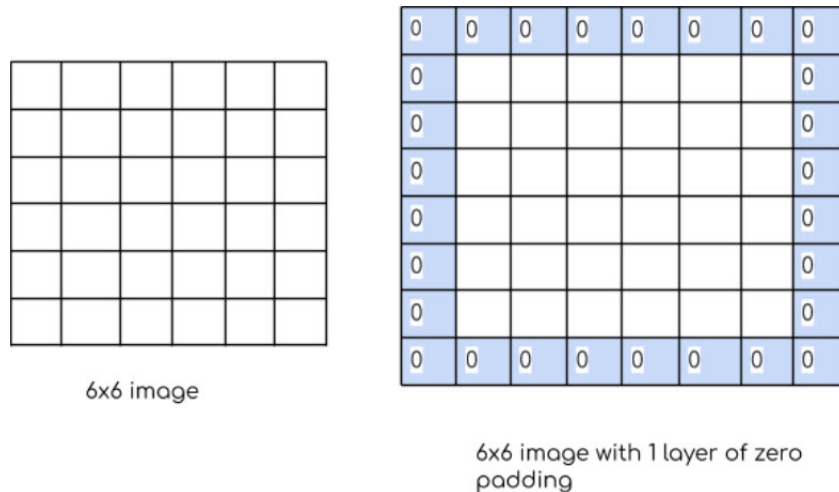
- Stride giúp ta xác định bao nhiêu pixel bước đi mỗi lần trượt bộ lọc tích chập trên ma trận đầu vào.
- Ví dụ khi stride là 1 tức là di chuyển các kernel 1 pixel. Khi stride là 2 tức là di chuyển các kernel đi 2 pixel và tiếp tục như vậy.
- Ví dụ hình ảnh dưới đây mô tả lớp chập hoạt động với stride là 2.



## 2.3 Padding - Đường viền

- Để giải quyết một số trường hợp kernel không phù hợp với ảnh đầu vào vì sau mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận output - Y đều nhỏ hơn ma trận input - X. Ta chọn tăng kích thước ảnh bằng cách thêm padding. Tức là thêm giá trị 0 ở viền ngoài ma trận X.

- Ví dụ dưới đây là thêm giá trị 0 ở viền với padding = 1.



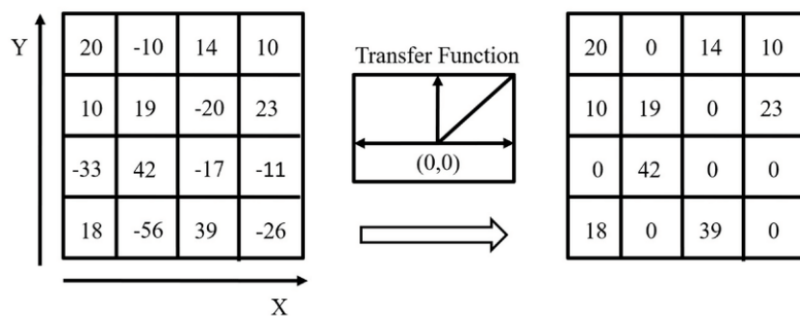
- Phép tính này được gọi là convolution với padding=1. Khi padding = k nghĩa là ta thêm k vector 0 vào mỗi phía của ma trận.

## 2.4 Rectified Linear Unit (Relu) - Hàm phi tuyến

- Giúp mạng neural học các đặc trưng tốt hơn và tăng tốc quá trình hội tụ trong quá trình huấn luyện.

- Giảm thiểu hiện tượng Gradient Vanishing (tức là khi gradient giảm đáng kể đến mức rất gần bằng 0 làm các trọng số của mạng neuron được cập nhật rất chậm hoặc thậm chí không cập nhật được nữa).

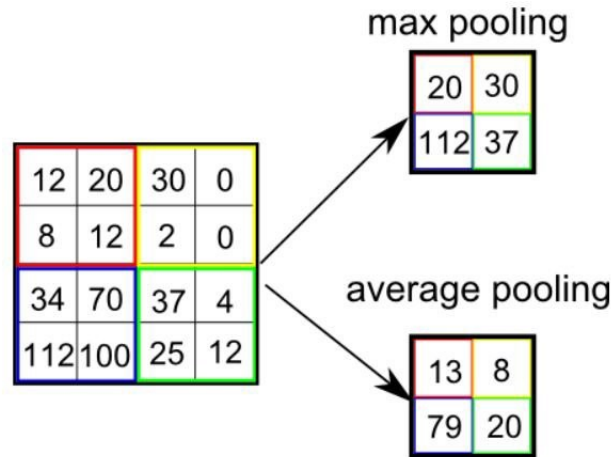
- Đầu ra  $f(x) = \max(0, x)$ . Tức là ma trận sau khi thực hiện phép tính chập và áp dụng hàm Relu ta sẽ ra được một ma trận không âm



- Tuy nhiên các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng này gọi là “Dying ReLU”. Khi các node bị chuyển thành 0 sẽ không còn ý nghĩa với linear activation ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent (vì bằng 0 nên không thể gradient được).  $\Rightarrow$  Leaky ReLU ra đời.

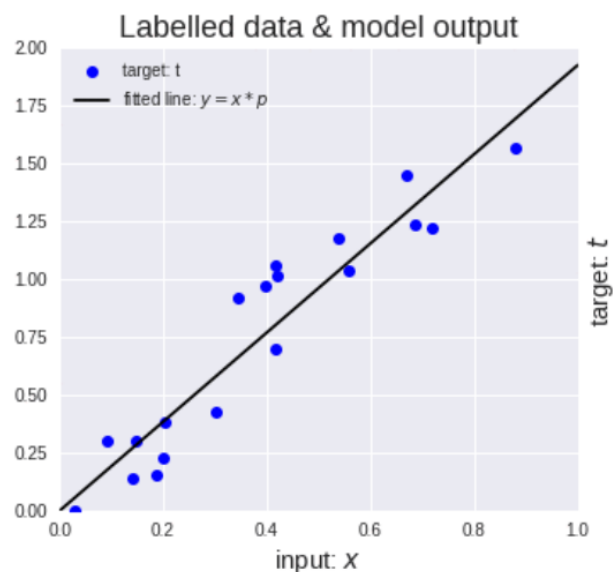
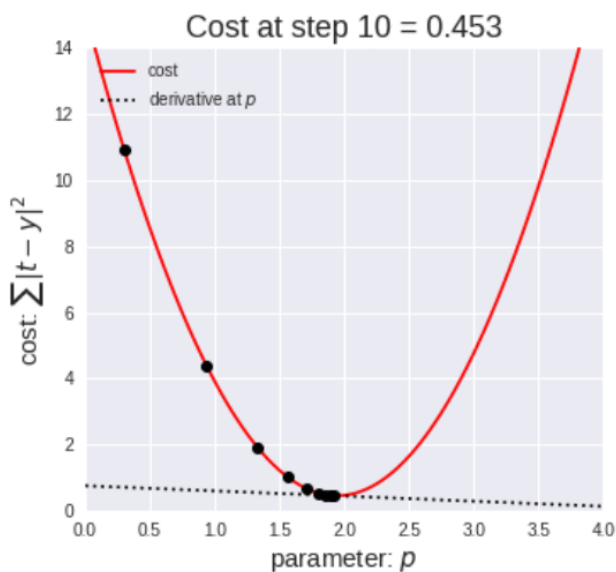
## 2.5 Pooling Layer - Lớp gộp

- Để giảm bớt kích thước đầu vào và trích xuất các đặc trưng quan trọng.
- Thường được dùng giữa các convolutional layer.
- Có 2 loại pooling layer phổ biến là max pooling và average pooling:
  - + Max pooling là pooling layer sẽ chọn ra giá trị lớn nhất trong mỗi khối đó để đưa ra đầu ra.
  - + Average pooling là pooling layer sẽ lấy giá trị trung bình trong mỗi khối để đưa ra đầu ra.



## 2.6 Gradient Descent

- Là thuật toán lặp tối ưu (iterative optimization algorithm) được sử dụng trong machine learning để tìm kết quả tốt nhất (minima of a curve). [6]
- Thuật toán sẽ lặp đi lặp lại nhiều lần đến khi nào giá trị **cost**  $\leq \epsilon$  (epsilon) đồng nghĩa với việc mô hình hội tụ. [7]

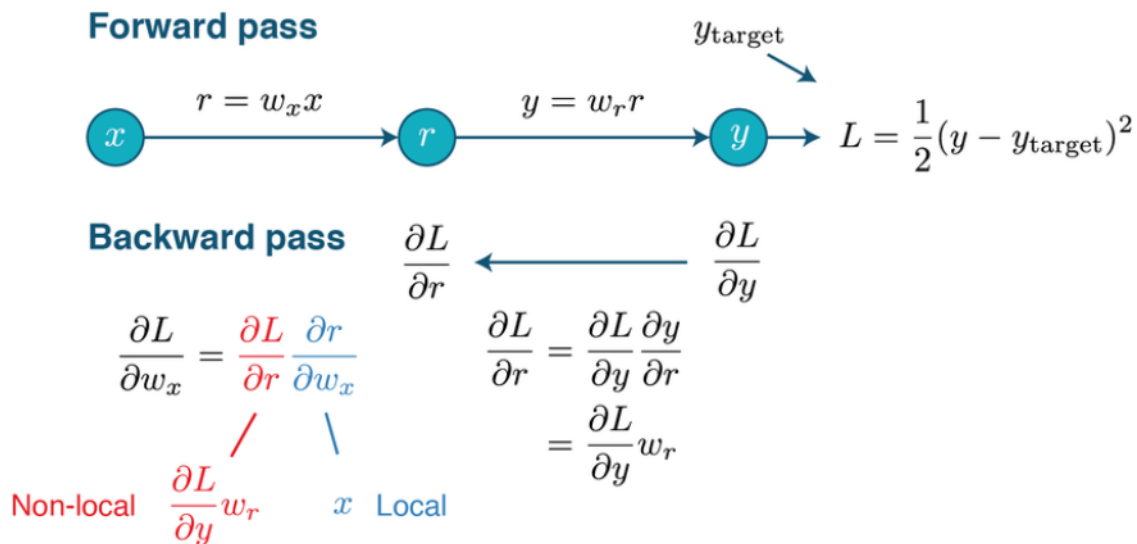


Gradient descent optimisation.

## 2.7 Forward Pass and Backward Pass - chạy thuận và chạy nghịch

- Forward Pass là quá trình đưa dữ liệu đầu vào đi qua một mạng neural network, các phép tính được thực hiện từ lớp đầu vào đến lớp đầu ra và cho ra kết quả đầu ra.

- Backward pass là quá trình tính toán đạo hàm (gradient) của hàm mất mát theo các tham số của mạng neural network. Quá trình này được thực hiện bằng thuật toán backpropagation, từ lớp đầu ra trở về lớp đầu vào, giúp cho việc cập nhật các trọng số mạng để tối ưu hoá mô hình được thực hiện một cách hiệu quả.

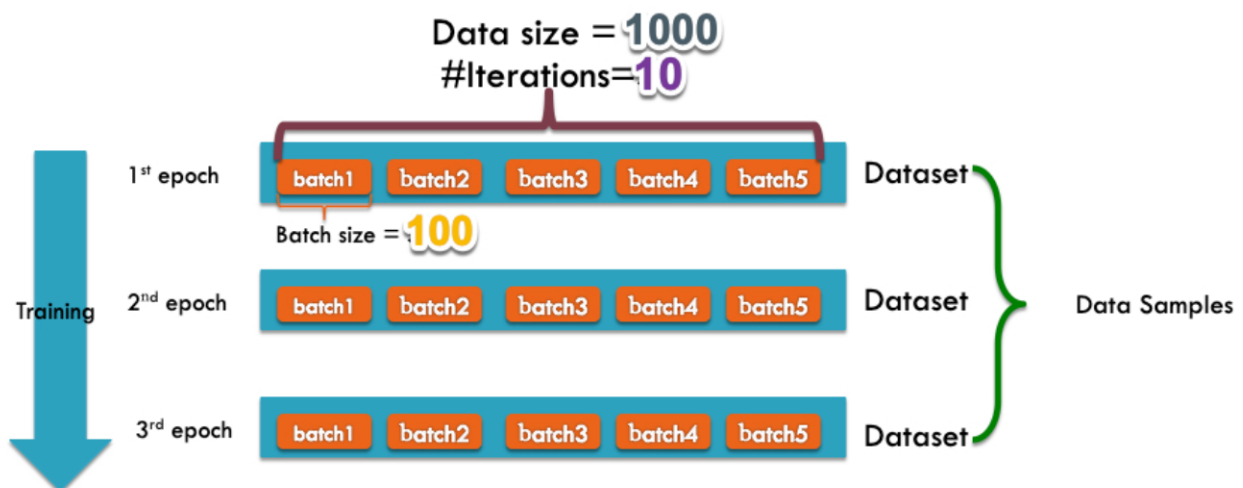


## 2.8 Batch size, Iteration, Epoch

Ở đây ta sẽ ví dụ chúng ta có một tập dữ liệu (dataset) 1000 tấm ảnh:

- **Batch size = 100**: nghĩa là ta sẽ lấy 100 tấm ảnh từ tập dữ liệu
- **Iteration (mini batch)**: nghĩa là số lần batch để lấy hết 1000 tấm ảnh. Như ở trên  

$$\text{batch} = 100 \text{ thì số Iteration} = \frac{1000}{100} = 10$$
- **Epoch**: nghĩa là số lần ta lấy lại tập dữ liệu đó để huấn luyện cho thuật toán hội tụ.



### 3 Implementation - Thực hiện

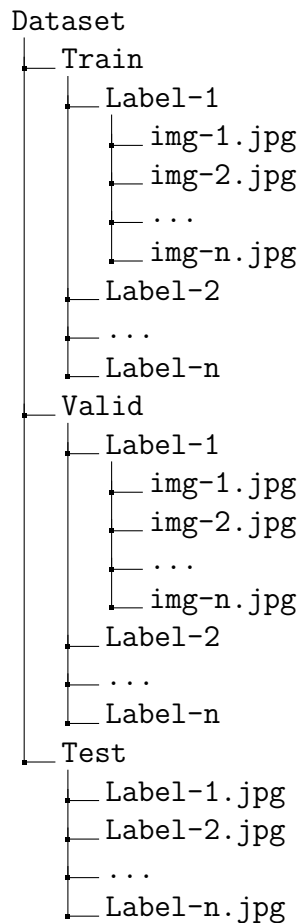
#### Các bước thực hiện

- + **Step 1:** Thu thập dữ liệu
- + **Step 2:** Xử lý dữ liệu
- + **Step 3:** Xây dựng model CNN
- + **Step 4:** Viết hàm loss đánh giá
- + **Step 5:** Nhập ảnh đầu vào từ tập test và demo kết quả

#### Công việc đã thực hiện

- ☒ Xây dựng model CNN với frame-work.
  - ☐ Pytorch
  - ☒ TensorFlow/Keras
  - ☐ PytorchLightning
- ☒ Demo kết quả.
  - ☒ Demo kết quả trên ảnh
  - ☐ Demo kết quả trên video

#### Cấu trúc dataset chung.





### 3.1 Recognition of all members of class from face images (you collected)

Code đầy đủ ở đây [8]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
                                Normalization, LeakyReLU, ReLU,
                                Dropout
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator
```

- Tiếp theo ta sẽ tiến hành thiết lập các siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- rescale=1./255: Sẽ chia giá trị pixel của ảnh cho 255 để đưa về khoảng giá trị từ 0 đến 1.
- shear range=0.2: Là tham số để làm xéo ảnh (shearing) với mức độ tối đa là 0.2.
- zoom range=0.2: Là tham số để zoom ảnh với mức độ tối đa là 0.2
- class mode='categorical': Là chúng ta tiến hành chuyển output về dạng one-hot encoding
- target size = (32,32): Là chúng ta sẽ tiến hành resize về kích thước ảnh về 32x32 pixel trước khi được đưa vào mô hình.

```
In [2]: # hyperparameter
batch_size = 128
epochs = 12
class_img = ['Cao Minh Quan', 'Huynh Anh DUY', 'Le Tri Dung', '
              Nguyen Hai Hoang', 'Nguyen
              Ngoc Nhan']

load_dataset = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

train_loader=load_dataset.flow_from_directory('/kaggle/input/face-
class/Face/Train',target_size=(
32,32), batch_size=128,
class_mode='categorical')

valid_loader=load_dataset.flow_from_directory('/kaggle/input/face-
class/Face/Val',target_size=(
32,32), batch_size=128,
class_mode='categorical')

train_loader.class_indices
```

Out [2]:Found 228 images belonging to 5 classes.

Found 51 images belonging to 5 classes.

```
{'CaoMinhQuan': 0,
```

```

'HuynhAnhDUy': 1,
'LeTriDung': 2,
'NguyenHaiHoang': 3,
'NguyenNgocNhan': 4}

```

```

In [3]: model = Sequential()
        model.add(Conv2D(32,(3,3), padding = 'same', input_shape=(32,32,3),
                           strides=(2,2)))

        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(64, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(128, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Flatten())
        model.add(Dense(124, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(32, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(5, activation='softmax'))
        model.summary()

```

Out [3]:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 16, 32)	896
leaky_re_lu (LeakyReLU)	(None, 16, 16, 32)	0
max_pooling2d (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_1 (Conv2D)	(None, 8, 8, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856

leaky_re_lu_2 (LeakyReLU)	(None, 4, 4, 128)	0
max_pooling2d_2 (MaxPooling 2D)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 124)	63612
dropout (Dropout)	(None, 124)	0
dense_1 (Dense)	(None, 32)	4000
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 5)	165

```
=====
Total params: 161,025
Trainable params: 161,025
Non-trainable params: 0
-----
```

- Tổng có 161,025 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model
- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```
In [4]: sgd = SGD(lr = 0.0001, decay = 1e-4, momentum = 0.9, nesterov= True)
        model.compile(loss= 'categorical_crossentropy', optimizer= Adam(),
                      metrics = ['accuracy'])
        train = model.fit(train_loader, validation_data=valid_loader,
                          batch_size=batch_size, epochs=
                          epochs, verbose=1)
```

Out[4]:

Epoch 1/12

2/2 [=====] - 8s 3s/step

- loss: 1.6038 - accuracy: 0.2675 - val\_loss: 1.5646 - val\_accuracy: 0.2941

Epoch 2/12

2/2 [=====] - 6s 3s/step

- loss: 1.5579 - accuracy: 0.3026 - val\_loss: 1.5153 - val\_accuracy: 0.3137

Epoch 3/12

2/2 [=====] - 6s 4s/step

```

- loss: 1.4965 - accuracy: 0.3465 - val_loss: 1.4347 - val_accuracy: 0.4902
Epoch 4/12
2/2 [=====] - 6s 4s/step
- loss: 1.4115 - accuracy: 0.4518 - val_loss: 1.2981 - val_accuracy: 0.6078
Epoch 5/12
2/2 [=====] - 6s 4s/step
- loss: 1.3176 - accuracy: 0.4781 - val_loss: 1.1729 - val_accuracy: 0.6667
Epoch 6/12
2/2 [=====] - 6s 3s/step
- loss: 1.2288 - accuracy: 0.5526 - val_loss: 1.0437 - val_accuracy: 0.7647
Epoch 7/12
2/2 [=====] - 6s 3s/step
- loss: 1.1006 - accuracy: 0.6096 - val_loss: 0.8703 - val_accuracy: 0.8235
Epoch 8/12
2/2 [=====] - 6s 4s/step
- loss: 0.9732 - accuracy: 0.7193 - val_loss: 0.7458 - val_accuracy: 0.8039
Epoch 9/12
2/2 [=====] - 6s 4s/step
- loss: 0.8790 - accuracy: 0.7675 - val_loss: 0.7358 - val_accuracy: 0.7647
Epoch 10/12
2/2 [=====] - 6s 3s/step
- loss: 0.7355 - accuracy: 0.7895 - val_loss: 0.4847 - val_accuracy: 0.8627
Epoch 11/12
2/2 [=====] - 6s 3s/step
- loss: 0.7038 - accuracy: 0.7719 - val_loss: 0.5246 - val_accuracy: 0.7647
Epoch 12/12
2/2 [=====] - 6s 4s/step
- loss: 0.4978 - accuracy: 0.8991 - val_loss: 0.3950 - val_accuracy: 0.8235

```

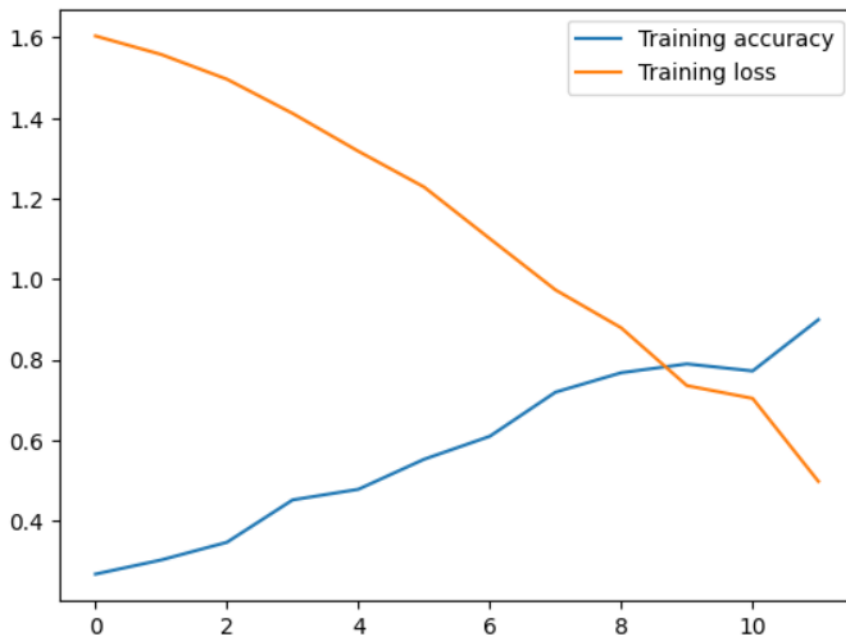
- Sau 12 epoch độ chính xác lên tới 86% đánh giá này giữa trên tập validation.
- Nhìn vào đồ thị dưới ta có thể thấy mô hình hội tụ khá nhanh sau 12 epoch

```

In [5]: accuracy = train.history['accuracy']
        loss = train.history['loss']
        plt.plot(accuracy, label='Training accuracy')
        plt.plot(loss, label='Training loss')
        plt.legend()
        plt.show()

```

Out [5]:



- Tiếp theo ta sẽ tiến hành lưu và load model

```
In [6]: # save model
        model.save("Face.h5")

        # Load model
        model_load=load_model('Face.h5')
```

- Cuối cùng ta sẽ tiến hành thử nghiệm model với ảnh từ tập test

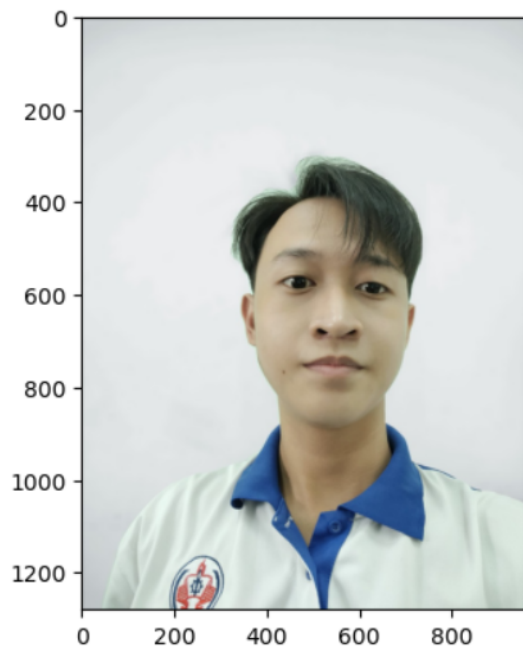
```
In [7]: import PIL.Image as Image
        from keras.utils import load_img, img_to_array

        url = '/kaggle/input/face-class/Face/Test/NguyenNgocNhan.jpg'
        # Load image
        img = Image.open(url)

        # Show image
        plt.imshow(img)
        plt.show()

        img=load_img(url,target_size=(32,32))
        img=img_to_array(img)
        img=img.astype('float32')
        img=img/255
        img=np.expand_dims(img,axis=0)

        result=(model_load.predict(img).argmax())
        print(class_img[result])
```



Out [7]: 1/1 [=====] - 0s 118ms/step  
Nguyen Ngoc Nhan

### 3.2 5 kinds of flowers (rose, lotus, water lily, apricot, daisy, pink)

Code đầy đủ ở đây [9]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
                        Normalization, LeakyReLU, ReLU,
                        Dropout
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator
```

- Tiếp theo ta sẽ tiến hành thiết lập các siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- rescale=1./255: Sẽ chia giá trị pixel của ảnh cho 255 để đưa về khoảng giá trị từ 0 đến 1.
- shear range=0.2: Là tham số để làm xéo ảnh (shearing) với mức độ tối đa là 0.2.
- zoom range=0.2: Là tham số để zoom ảnh với mức độ tối đa là 0.2
- class mode='categorical': Là chúng ta tiến hành chuyển output về dạng one-hot encoding
- target size = (86,86): Là chúng ta sẽ tiến hành resize về kích thước ảnh về 86x86 pixel trước khi được đưa vào mô hình.

```
In [2]: ## hyperparameter
batch_size = 128
epochs = 60
class_img = ['Apricots', 'Daisy', 'Lotus', 'Pink', 'Waterlily']
load_dataset = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

train_loader=load_dataset.flow_from_directory('/kaggle/input/flower-
s/FLOWERS/Train',target_size=(
86,86), batch_size=128,
class_mode='categorical')
valid_loader=load_dataset.flow_from_directory('/kaggle/input/flower-
s/FLOWERS/Train',target_size=(
86,86), batch_size=128,
class_mode='categorical')

train_loader.class_indices
```

Found 297 images belonging to 5 classes.

Found 297 images belonging to 5 classes.

Out [2]:{'Apricots': 0, 'Daisy': 1, 'Lotus': 2, 'Pink': 3, 'Waterlily': 4}

```
In [3]: model = Sequential()
        model.add(Conv2D(32,(3,3), padding = 'same', input_shape=(86,86,3),
                           strides=(2,2)))

        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(64, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(128, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Flatten())
        model.add(Dense(124, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(32, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(5, activation='softmax'))
        model.summary()
```

Out [3]:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 43, 43, 32)	896
leaky_re_lu (LeakyReLU)	(None, 43, 43, 32)	0
max_pooling2d (MaxPooling2D)	(None, 21, 21, 32)	0
conv2d_1 (Conv2D)	(None, 21, 21, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 21, 21, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 10, 10, 64)	0
conv2d_2 (Conv2D)	(None, 10, 10, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 10, 10, 128)	0



max_pooling2d_2 (MaxPooling 2D)	(None, 5, 5, 128)	0
flatten (Flatten)	(None, 3200)	0
dense (Dense)	(None, 124)	396924
dropout (Dropout)	(None, 124)	0
dense_1 (Dense)	(None, 32)	4000
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 5)	165

```

=====
Total params: 494,337
Trainable params: 494,337
Non-trainable params: 0
-----

```

- Tổng có 494,337 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model
- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```

In [4]: sgd = SGD(lr = 0.0001, decay = 1e-4, momentum = 0.7, nesterov= True)
        model.compile(loss= 'categorical_crossentropy', optimizer= Adam(),
                       metrics = ['accuracy'])
        train = model.fit(train_loader, validation_data=valid_loader,
                           batch_size=batch_size, epochs=
                           epochs, verbose=1)

```

Out[4]:

Epoch 1/60

3/3 [=====] - 9s 4s/step

- loss: 1.5302 - accuracy: 0.2862 - val\_loss: 1.4407 - val\_accuracy: 0.3636

Epoch 2/60

3/3 [=====] - 6s 2s/step

- loss: 1.4444 - accuracy: 0.3333 - val\_loss: 1.3670 - val\_accuracy: 0.4040

Epoch 3/60

3/3 [=====] - 6s 3s/step

- loss: 1.4057 - accuracy: 0.3502 - val\_loss: 1.3413 - val\_accuracy: 0.3636

Epoch 4/60

```

3/3 [=====] - 6s 3s/step
- loss: 1.3683 - accuracy: 0.3838 - val_loss: 1.2755 - val_accuracy: 0.4680
Epoch 5/60
3/3 [=====] - 6s 3s/step
- loss: 1.3254 - accuracy: 0.4007 - val_loss: 1.2430 - val_accuracy: 0.4310
...
Epoch 56/60
3/3 [=====] - 6s 2s/step
- loss: 0.3366 - accuracy: 0.8687 - val_loss: 0.2794 - val_accuracy: 0.8923
Epoch 57/60
3/3 [=====] - 6s 3s/step
- loss: 0.3818 - accuracy: 0.8350 - val_loss: 0.2783 - val_accuracy: 0.9024
Epoch 58/60
3/3 [=====] - 6s 3s/step
- loss: 0.3182 - accuracy: 0.8855 - val_loss: 0.2602 - val_accuracy: 0.9091
Epoch 59/60
3/3 [=====] - 6s 2s/step
- loss: 0.3696 - accuracy: 0.8384 - val_loss: 0.2401 - val_accuracy: 0.9158
Epoch 60/60
3/3 [=====] - 6s 2s/step
- loss: 0.3139 - accuracy: 0.8788 - val_loss: 0.2824 - val_accuracy: 0.9024

```

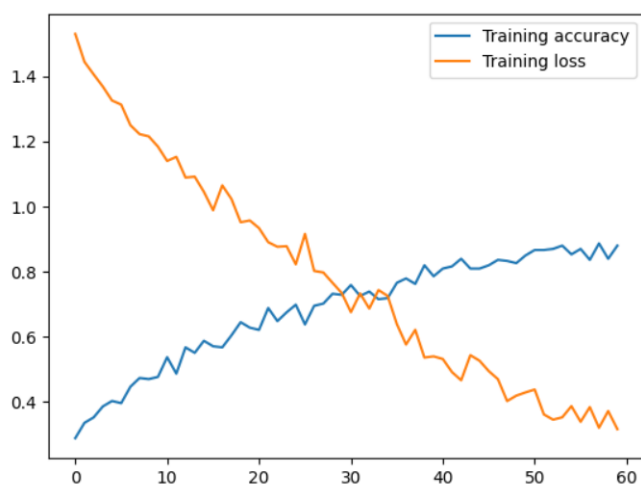
- Sau 60 epoch độ chính xác lên tới 91% đánh giá này giữa trên tập validation.
- Nhìn vào đồ thị dưới ta có thể thấy mô hình hội tụ khá nhanh sau 60 epoch

```

In [5]: accuracy = train.history['accuracy']
        loss = train.history['loss']
        plt.plot(accuracy, label='Training accuracy')
        plt.plot(loss, label='Training loss')
        plt.legend()
        plt.show()

```

Out [5]:



- Tiếp theo ta sẽ tiến hành lưu và load model

```
In [6]: # save model
        model.save("Flower.h5")

        # Load model
        model_load=load_model('Flower.h5')
```

- Cuối cùng ta sẽ tiến hành thử nghiệm model với ảnh từ tập test

```
In [7]: import PIL.Image as Image
        from keras.utils import load_img, img_to_array

        url = '/kaggle/input/flower-s/FLOWERS/Test/WATERLILY2.jpg'
        # Load img
        img = Image.open(url)

        # Show img
        plt.imshow(img)
        plt.show()

        img=load_img(url,target_size=(86,86))
        img=img_to_array(img)
        img=img.astype('float32')
        img=img/255
        img=np.expand_dims(img,axis=0)

        result=(model_load.predict(img).argmax())
        print(class_img[result])
```



```
Out [7]: 1/1 [=====] - 0s 24ms/step
        Waterlily
```

### 3.3 10 Vietnamese dishes (bún bò, chè, bánh xèo.....)

#### Code đầy đủ ở đây [10]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
                        Normalization, LeakyReLU, ReLU,
                        Dropout
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator
```

- Tiếp theo ta sẽ tiến hành thiết lập các siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- rescale=1./255: Sẽ chia giá trị pixel của ảnh cho 255 để đưa về khoảng giá trị từ 0 đến 1.
- shear range=0.2: Là tham số để làm xéo ảnh (shearing) với mức độ tối đa là 0.2.
- zoom range=0.2: Là tham số để zoom ảnh với mức độ tối đa là 0.2
- class mode='categorical': Là chúng ta tiến hành chuyển output về dạng one-hot encoding
- target size = (244,244): Là chúng ta sẽ tiến hành resize về kích thước ảnh về 244x244 pixel trước khi được đưa vào mô hình.

```
In [2]: # hyperparameter
batch_size = 128
epochs = 40
class_img = ['Banh bot loc', 'Banh chung', 'Banh khot', 'Banh mi', '
            'Banh trang nuong', 'Bun bo Hue',
            'Bun thit nuong',
            'Bun thit nuong', 'Chao long', 'Com tam', 'Xoi xeo']
load_dataset = ImageDataGenerator(rescale = 1./255,
                                shear_range = 0.2,
                                zoom_range = 0.2,
                                horizontal_flip = True)
train_loader=load_dataset.flow_from_directory('/kaggle/input/vn-food
            /Food/Train',target_size=(244,
            244), batch_size=128,
            class_mode='categorical')
valid_loader=load_dataset.flow_from_directory('/kaggle/input/vn-food
            /Food/Val',target_size=(244,
            244), batch_size=128,
            class_mode='categorical')

train_loader.class_indices
```

Found 332 images belonging to 10 classes.

Found 124 images belonging to 10 classes.

```
Out [2]:{'Banh bot loc': 0, 'Banh chung': 1, 'Banh khot': 2, 'Banh mi': 3,
'Banh trang nuong': 4, 'Bun bo Hue': 5, 'Bun thit nuong': 6,
'Chao long': 7, 'Com tam': 8, 'Xoi xeo': 9}
```

```
In [3]: model = Sequential()
        model.add(Conv2D(32, (3,3), padding = 'same', input_shape=(244,244,3)
                        , strides=(2,2)))

        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(64, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(128, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Flatten())
        model.add(Dense(124, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(32, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(10, activation='softmax'))
        model.summary()
```

Out [3]:

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
=====		
conv2d_3 (Conv2D)	(None, 122, 122, 32)	896
leaky_re_lu_3 (LeakyReLU)	(None, 122, 122, 32)	0
max_pooling2d_3 (MaxPooling 2D)	(None, 61, 61, 32)	0
conv2d_4 (Conv2D)	(None, 61, 61, 64)	18496
leaky_re_lu_4 (LeakyReLU)	(None, 61, 61, 64)	0
max_pooling2d_4 (MaxPooling 2D)	(None, 30, 30, 64)	0
conv2d_5 (Conv2D)	(None, 30, 30, 128)	73856
leaky_re_lu_5 (LeakyReLU)	(None, 30, 30, 128)	0
max_pooling2d_5 (MaxPooling 2D)	(None, 15, 15, 128)	0

2D)

flatten_1 (Flatten)	(None, 28800)	0
dense_3 (Dense)	(None, 124)	3571324
dropout_2 (Dropout)	(None, 124)	0
dense_4 (Dense)	(None, 32)	4000
dropout_3 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 10)	330

```
=====
Total params: 3,668,902
Trainable params: 3,668,902
Non-trainable params: 0
-----
```

- Tổng có 3,668,902 tham số
  - Tiếp theo ta sẽ tiến hành train và đánh giá model
  - Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting
- [5]

```
In [4]: sgd = SGD(lr = 0.0001, decay = 1e-4, momentum = 0.9, nesterov= True)
        model.compile(loss= 'categorical_crossentropy', optimizer= Adam(),
                      metrics = ['accuracy'])
        train = model.fit(train_loader, validation_data=valid_loader,
                          batch_size=batch_size, epochs=
                          epochs, verbose=1)
```

Out[4]:

Epoch 1/40

3/3 [=====] - 13s 5s/step

- loss: 2.4524 - accuracy: 0.0813 - val\_loss: 2.2933 - val\_accuracy: 0.1371

Epoch 2/40

3/3 [=====] - 10s 4s/step

- loss: 2.3231 - accuracy: 0.1265 - val\_loss: 2.2725 - val\_accuracy: 0.2097

Epoch 3/40

3/3 [=====] - 10s 4s/step

- loss: 2.2847 - accuracy: 0.1536 - val\_loss: 2.2551 - val\_accuracy: 0.1774

Epoch 4/40

3/3 [=====] - 10s 3s/step

```

- loss: 2.3011 - accuracy: 0.1627 - val_loss: 2.2677 - val_accuracy: 0.1452
Epoch 5/40
3/3 [=====] - 10s 4s/step
- loss: 2.2920 - accuracy: 0.1295 - val_loss: 2.2501 - val_accuracy: 0.1694
...
Epoch 36/40
3/3 [=====] - 10s 4s/step
- loss: 0.5336 - accuracy: 0.8253 - val_loss: 0.6691 - val_accuracy: 0.8387
Epoch 37/40
3/3 [=====] - 10s 3s/step
- loss: 0.4436 - accuracy: 0.8494 - val_loss: 0.6373 - val_accuracy: 0.8306
Epoch 38/40
3/3 [=====] - 10s 3s/step
- loss: 0.5023 - accuracy: 0.8373 - val_loss: 0.7376 - val_accuracy: 0.8065
Epoch 39/40
3/3 [=====] - 10s 4s/step
- loss: 0.3306 - accuracy: 0.8855 - val_loss: 0.7848 - val_accuracy: 0.8226
Epoch 40/40
3/3 [=====] - 10s 4s/step
- loss: 0.4333 - accuracy: 0.8434 - val_loss: 0.7471 - val_accuracy: 0.8306

```

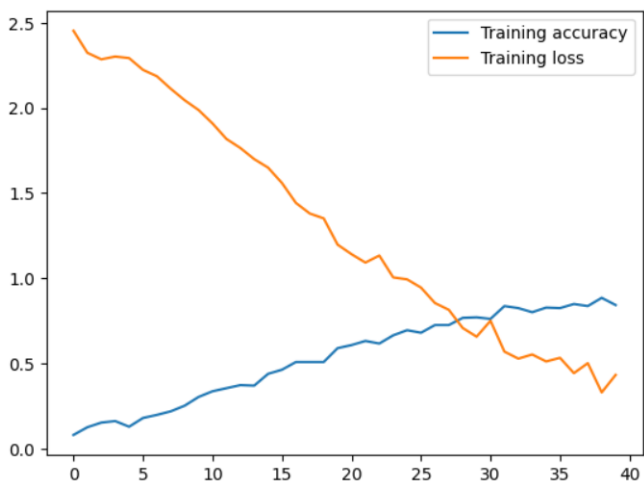
- Sau 40 epoch độ chính xác lên tới 83% đánh giá này giữa trên tập validation.
- Nhìn vào đồ thị dưới ta có thể thấy mô hình hội tụ khá nhanh sau 40 epoch

```

In [5]: accuracy = train.history['accuracy']
        loss = train.history['loss']
        plt.plot(accuracy, label='Training accuracy')
        plt.plot(loss, label='Training loss')
        plt.legend()
        plt.show()

```

Out [5]:



- Tiếp theo ta sẽ tiến hành lưu và load model

```
In [6]: # save model
        model.save("Food.h5")

        # Load model
        model_load=load_model('Food.h5')
```

- Cuối cùng ta sẽ tiến hành thử nghiệm model với ảnh từ tập test

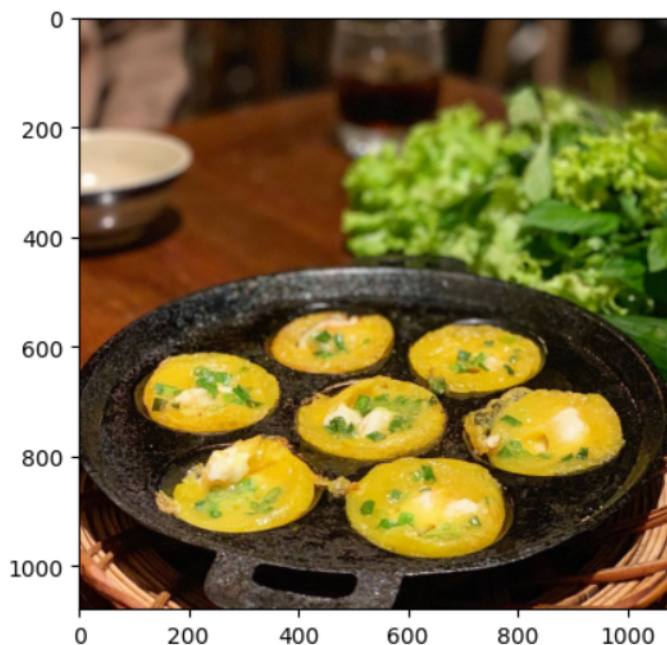
```
In [7]: import PIL.Image as Image
        from keras.utils import load_img, img_to_array

        url = '/kaggle/input/vn-food/Food/Test/BanhKhot.jpg'
        # Load img
        img = Image.open(url)

        # sShow img
        plt.imshow(img)
        plt.show()

        img=load_img(url,target_size=(244,244))
        img=img_to_array(img)
        img=img.astype('float32')
        img=img/255
        img=np.expand_dims(img,axis=0)

        result=(model_load.predict(img).argmax())
        print(class_img[result])
```



```
Out [7]: 1/1 [=====] - 0s 125ms/step
        Banh khot
```



### 3.4 VN banknotes (5000vnd, 10000vnd, 20k, 50k, 100k, 500k)

Code đầy đủ ở đây [11]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
                        Normalization, LeakyReLU, ReLU,
                        Dropout
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator
```

- Tiếp theo ta sẽ tiến hành thiết lập các siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- rescale=1./255: Sẽ chia giá trị pixel của ảnh cho 255 để đưa về khoảng giá trị từ 0 đến 1.
- shear range=0.2: Là tham số để làm xéo ảnh (shearing) với mức độ tối đa là 0.2.
- zoom range=0.2: Là tham số để zoom ảnh với mức độ tối đa là 0.2
- class mode='categorical': Là chúng ta tiến hành chuyển output về dạng one-hot encoding
- target size = (128,128): Là chúng ta sẽ tiến hành resize về kích thước ảnh về 128x128 pixel trước khi được đưa vào mô hình.

```
In [2]: # hyperparameter
batch_size = 128
epochs = 80
class_img = ['100k', '10k', '20k', '500k', '50k', '5k']

load_dataset = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

train_loader=load_dataset.flow_from_directory('/kaggle/input/vn-
money/Money/Train',target_size
=(128,128), batch_size=128,
class_mode='categorical')
valid_loader=load_dataset.flow_from_directory('/kaggle/input/vn-
money/Money/Val',target_size=(
128,128), batch_size=128,
class_mode='categorical')

train_loader.class_indices
```

Found 355 images belonging to 6 classes.

Found 12 images belonging to 6 classes.

Out [2]:{'100k': 0, '10k': 1, '20k': 2, '500k': 3, '50k': 4, '5k': 5}

```
In [3]: model = Sequential()
        model.add(Conv2D(32,(3,3), padding = 'same', input_shape=(128,128,3)
                        , strides= (2,2)))

        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(64, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(128, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Flatten())
        model.add(Dense(124, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(32, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(6, activation='softmax'))
        model.summary()
```

Out [3]:

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
=====		
conv2d_6 (Conv2D)	(None, 64, 64, 32)	896
leaky_re_lu_6 (LeakyReLU)	(None, 64, 64, 32)	0
max_pooling2d_6 (MaxPooling 2D)	(None, 32, 32, 32)	0
conv2d_7 (Conv2D)	(None, 32, 32, 64)	18496
leaky_re_lu_7 (LeakyReLU)	(None, 32, 32, 64)	0
max_pooling2d_7 (MaxPooling 2D)	(None, 16, 16, 64)	0
conv2d_8 (Conv2D)	(None, 16, 16, 128)	73856
leaky_re_lu_8 (LeakyReLU)	(None, 16, 16, 128)	0
max_pooling2d_8 (MaxPooling 2D)	(None, 8, 8, 128)	0

2D)

flatten_2 (Flatten)	(None, 8192)	0
dense_6 (Dense)	(None, 124)	1015932
dropout_4 (Dropout)	(None, 124)	0
dense_7 (Dense)	(None, 32)	4000
dropout_5 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 6)	198

```
=====
Total params: 1,113,378
Trainable params: 1,113,378
Non-trainable params: 0
-----
```

- Tổng có 1,113,378 tham số
  - Tiếp theo ta sẽ tiến hành train và đánh giá model
  - Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting
- [5]

```
In [4]: sgd = SGD(lr = 0.0001, decay = 1e-4, momentum = 0.9, nesterov= True)
        model.compile(loss= 'categorical_crossentropy', optimizer= Adam(),
                      metrics = ['accuracy'])
        train = model.fit(train_loader, validation_data=valid_loader,
                          batch_size=batch_size, epochs=
                          epochs, verbose=1)
```

Out[4]:

Epoch 1/80

3/3 [=====] - 5s 1s/step  
- loss: 1.4622 - accuracy: 0.4845 - val\_loss: 2.0072 - val\_accuracy: 0.1667

Epoch 2/80

3/3 [=====] - 4s 1s/step  
- loss: 1.1466 - accuracy: 0.6113 - val\_loss: 2.3326 - val\_accuracy: 0.1667

Epoch 3/80

3/3 [=====] - 4s 1s/step  
- loss: 0.9976 - accuracy: 0.7352 - val\_loss: 2.5271 - val\_accuracy: 0.1667

Epoch 4/80

3/3 [=====] - 4s 1s/step

```

- loss: 0.8540 - accuracy: 0.7465 - val_loss: 1.8140 - val_accuracy: 0.1667
Epoch 5/80
3/3 [=====] - 4s 1s/step
- loss: 0.7493 - accuracy: 0.7577 - val_loss: 1.8497 - val_accuracy: 0.1667
...
Epoch 76/80
3/3 [=====] - 4s 1s/step
- loss: 0.0225 - accuracy: 0.9972 - val_loss: 0.4236 - val_accuracy: 0.9167
Epoch 77/80
3/3 [=====] - 4s 1s/step
- loss: 0.0454 - accuracy: 0.9803 - val_loss: 1.5360 - val_accuracy: 0.6667
Epoch 78/80
3/3 [=====] - 4s 1s/step
- loss: 0.0358 - accuracy: 0.9859 - val_loss: 1.4227 - val_accuracy: 0.8333
Epoch 79/80
3/3 [=====] - 4s 1s/step
- loss: 0.0432 - accuracy: 0.9859 - val_loss: 1.3544 - val_accuracy: 0.7500
Epoch 80/80
3/3 [=====] - 4s 1s/step
- loss: 0.0403 - accuracy: 0.9887 - val_loss: 1.0277 - val_accuracy: 0.8333

```

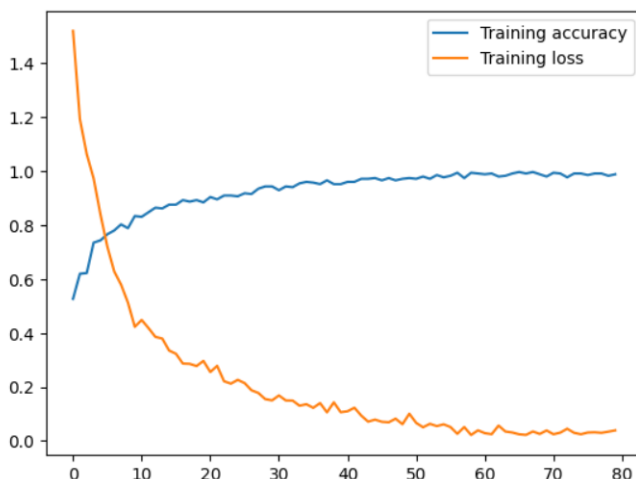
- Sau 80 epoch độ chính xác lên tới 91% đánh giá này giữa trên tập validation.
- Nhìn vào đồ thị dưới ta có thể thấy mô hình hội tụ khá nhanh sau 80 epoch

```

In [5]: accuracy = train.history['accuracy']
        loss = train.history['loss']
        plt.plot(accuracy, label='Training accuracy')
        plt.plot(loss, label='Training loss')
        plt.legend()
        plt.show()

```

Out [5]:



- Tiếp theo ta sẽ tiến hành lưu và load model

```
In [6]: # save model
        model.save("Money.h5")

        # Load model
        model_load=load_model('Money.h5')
```

- Cuối cùng ta sẽ tiến hành thử nghiệm model với ảnh từ tập test

```
In [7]: import PIL.Image as Image
        from keras.utils import load_img, img_to_array

        url = '/kaggle/input/vn-money/Money/Test/20k.jpg'

        img = Image.open(url)

        plt.imshow(img)
        plt.show()

        img=load_img(url,target_size=(128,128))
        img=img_to_array(img)
        img=img.astype('float32')
        img=img/255
        img=np.expand_dims(img,axis=0)

        result=(model_load.predict(img).argmax())
        print(class_img[result])
```



1/1 [=====] - 0s 26ms/step

Out [7]:20k

### 3.5 Prediction of your future based on hand palm outline or face or fingerprint.

Code đầy đủ ở đây [12]

- Đầu tiên ta sẽ import các thư viện cần thiết cho việc xây dựng và thực thi model

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
                                Normalization, LeakyReLU, ReLU,
                                Dropout
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator
```

- Tiếp theo ta sẽ tiến hành thiết lập các siêu tham số (hyperparameter) và chia dữ liệu theo batch cho model

- rescale=1./255: Sẽ chia giá trị pixel của ảnh cho 255 để đưa về khoảng giá trị từ 0 đến 1.
- shear range=0.2: Là tham số để làm xéo ảnh (shearing) với mức độ tối đa là 0.2.
- zoom range=0.2: Là tham số để zoom ảnh với mức độ tối đa là 0.2
- class mode='categorical': Là chúng ta tiến hành chuyển output về dạng one-hot encoding
- target size = (32,32): Là chúng ta sẽ tiến hành resize về kích thước ảnh về 32x32 pixel trước khi được đưa vào mô hình.

```
In [2]: # hyperparameter
batch_size = 128
epochs = 12
class_img = ['Tuong lai lam ca si', 'Tuong lai lam gamer', 'Tuong
            lai lam trader', 'Tuong lai
            lam ky su robot', 'Tuong lai
            lam ky su']

load_dataset = ImageDataGenerator(rescale = 1./255,
                                shear_range = 0.2,
                                zoom_range = 0.2,
                                horizontal_flip = True)

train_loader=load_dataset.flow_from_directory('/kaggle/input/face-
class/Face/Train',target_size=(
32,32), batch_size=128,
class_mode='categorical')

valid_loader=load_dataset.flow_from_directory('/kaggle/input/face-
class/Face/Val',target_size=(
32,32), batch_size=128,
class_mode='categorical')

train_loader.class_indices
```

Out [2]:Found 228 images belonging to 5 classes.  
Found 51 images belonging to 5 classes.

```
In [3]: model = Sequential()
        model.add(Conv2D(32,(3,3), padding = 'same', input_shape=(32,32,3),
                           strides=(2,2)))

        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(64, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Conv2D(128, (3,3), strides=(1,1), padding='same'))
        model.add(LeakyReLU())
        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

        model.add(Flatten())
        model.add(Dense(124, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(32, activation='relu'))
        model.add(Dropout(0.1))
        model.add(Dense(5, activation='softmax'))
        model.summary()
```

Out [3]:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 16, 32)	896
leaky_re_lu (LeakyReLU)	(None, 16, 16, 32)	0
max_pooling2d (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_1 (Conv2D)	(None, 8, 8, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 4, 4, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0

flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 124)	63612
dropout (Dropout)	(None, 124)	0
dense_1 (Dense)	(None, 32)	4000
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 5)	165

```

=====
Total params: 161,025
Trainable params: 161,025
Non-trainable params: 0
-----

```

- Tổng có 161,025 tham số
- Tiếp theo ta sẽ tiến hành train và đánh giá model
- Ở đây ta sử dụng Weight Decay (L2 Regularization) bằng 0.0001 để giảm thiểu Overfitting [5]

```

In [4]: sgd = SGD(lr = 0.0001, decay = 1e-4, momentum = 0.9, nesterov= True)
        model.compile(loss= 'categorical_crossentropy', optimizer= Adam(),
                      metrics = ['accuracy'])
        train = model.fit(train_loader, validation_data=valid_loader,
                          batch_size=batch_size, epochs=
                          epochs, verbose=1)

```

Out [4]:

Epoch 1/12

2/2 [=====] - 15s 4s/step

- loss: 1.6052 - accuracy: 0.2763 - val\_loss: 1.5621 - val\_accuracy: 0.2549

Epoch 2/12

2/2 [=====] - 6s 4s/step

- loss: 1.5600 - accuracy: 0.2632 - val\_loss: 1.5221 - val\_accuracy: 0.3922

Epoch 3/12

2/2 [=====] - 6s 3s/step

- loss: 1.5083 - accuracy: 0.3640 - val\_loss: 1.4633 - val\_accuracy: 0.4706

Epoch 4/12

2/2 [=====] - 6s 4s/step

- loss: 1.4557 - accuracy: 0.3772 - val\_loss: 1.3476 - val\_accuracy: 0.6275

Epoch 5/12

2/2 [=====] - 6s 3s/step



```

- loss: 1.3528 - accuracy: 0.5570 - val_loss: 1.2784 - val_accuracy: 0.7451
Epoch 6/12
2/2 [=====] - 6s 3s/step
- loss: 1.2618 - accuracy: 0.7018 - val_loss: 1.0963 - val_accuracy: 0.7647
Epoch 7/12
2/2 [=====] - 6s 3s/step
- loss: 1.0564 - accuracy: 0.8070 - val_loss: 0.9574 - val_accuracy: 0.7647
Epoch 8/12
2/2 [=====] - 6s 3s/step
- loss: 0.9627 - accuracy: 0.7675 - val_loss: 0.7836 - val_accuracy: 0.8627
Epoch 9/12
2/2 [=====] - 6s 4s/step
- loss: 0.7992 - accuracy: 0.7939 - val_loss: 0.6779 - val_accuracy: 0.7647
Epoch 10/12
2/2 [=====] - 6s 3s/step
- loss: 0.6521 - accuracy: 0.8421 - val_loss: 0.5592 - val_accuracy: 0.7647
Epoch 11/12
2/2 [=====] - 6s 3s/step
- loss: 0.5331 - accuracy: 0.8684 - val_loss: 0.4499 - val_accuracy: 0.8235
Epoch 12/12
2/2 [=====] - 6s 4s/step
- loss: 0.4666 - accuracy: 0.8684 - val_loss: 0.3116 - val_accuracy: 0.9216

```

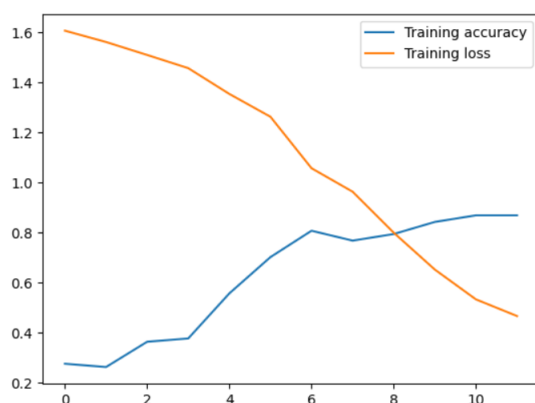
- Sau 12 epoch độ chính xác lên tới 92% đánh giá này giữa trên tập validation.
- Nhìn vào đồ thị dưới ta có thể thấy mô hình hội tụ khá nhanh sau 12 epoch

```

In [5]: accuracy = train.history['accuracy']
        loss = train.history['loss']
        plt.plot(accuracy, label='Training accuracy')
        plt.plot(loss, label='Training loss')
        plt.legend()
        plt.show()

```

Out [5]:



- Tiếp theo ta sẽ tiến hành lưu và load model

```
In [6]: # save model
        model.save("Future.h5")

        # Load model
        model_load=load_model('Future.h5')
```

- Cuối cùng ta sẽ tiến hành thử nghiệm model với ảnh từ tập test

```
In [7]: import PIL.Image as Image
        from keras.utils import load_img, img_to_array

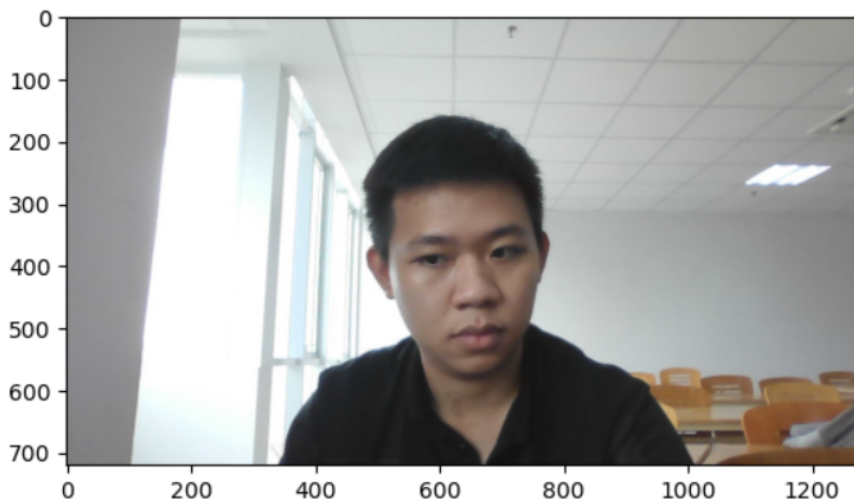
        url = '/kaggle/input/face-class/Face/Test/WIN_20230331_08_45_23_Pro.
                jpg'

        img = Image.open(url)

        plt.imshow(img)
        plt.show()

        img=load_img(url,target_size=(32,32))
        img=img_to_array(img)
        img=img.astype('float32')
        img=img/255
        img=np.expand_dims(img,axis=0)

        result=(model_load.predict(img).argmax())
        print(class_img[result])
```



1/1 [=====] - 0s 19ms/step

Out [7]:Tuong lai lam trader

## 4 Results - Kết quả

- Sau khi sử dụng CNN với độ chính xác trên 80% là một thành công hơn so với các mô hình machine learning trong việc xử lý ảnh và phân loại các đối tượng. Nó phụ thuộc rất nhiều vào các siêu tham số (hyperparameters) như epochs, batch size, dropout, weight decay, optimizer và cả độ đa dạng của data.
- Với CNN, chúng ta có thể tạo ra các mô hình phân loại ảnh chất lượng cao và đáng tin cậy, đặc biệt là trong các bài toán phân loại ảnh có tính chất phức tạp như phân loại hoa, phân loại động vật, nhận diện khuôn mặt, và nhiều bài toán khác. CNN đã trở thành một công cụ quan trọng trong lĩnh vực xử lý ảnh và thường được sử dụng để giải quyết các bài toán phân loại ảnh trong các ứng dụng thực tế.

## 5 Conclusions - Kết luận

- Convolutional neural network (CNN) là một loại mạng nơ-ron nhân tạo rất phổ biến trong việc xử lý ảnh và nhận dạng hình ảnh. Nó được thiết kế để phát hiện các đặc trưng của ảnh thông qua các lớp tích chập và kết hợp chúng lại để tạo ra một dự đoán cuối cùng.
- CNN đã đạt được những thành tựu lớn trong lĩnh vực nhận dạng hình ảnh, bao gồm nhận dạng khuôn mặt, phân loại hình ảnh, nhận dạng vật thể và nhiều ứng dụng khác. Nó cũng được sử dụng trong nhiều lĩnh vực khác nhau bao gồm xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói, xử lý tín hiệu và nhiều ứng dụng khác.
- Mặc dù CNN đã đạt được nhiều thành tựu đáng kể, nhưng vẫn còn nhiều thách thức cần giải quyết, bao gồm độ chính xác của mô hình, kích thước mô hình và thời gian huấn luyện. Tuy nhiên các pretrain model hiện nay thường được xây dựng trên mạng CNN và các biến thể của nó.

## 6 References

- [1] <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>
- [2] <https://nttuan8.com/bai-6-convolutional-neural-network/>
- [3] <https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2>
- [4] <https://aicurious.io/blog/2019-09-23-cac-ham-kich-hoat-activation-function-trong-neural-networks>
- [5] <https://machinelearningcoban.com/2017/03/04/overfitting/>
- [6] <https://www.phamduytung.com/blog/2018-10-02-understanding-epoch-batchsize-iterations/>
- [7] <https://medium.com/onfido-tech/machine-learning-101-be2e0a86c96a>

**GitHub:** <https://github.com/tooniesnguyen/>

- [8] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/face-detect-class-tf.ipynb>
- [9] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/flower-tf.ipynb>
- [10] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/predict-food-tf.ipynb>
- [11] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/predict-money-tf.ipynb>
- [12] <https://github.com/tooniesnguyen/AI-UTE/blob/main/Notebooks/MidtermProject/predict-future-tf.ipynb>

**TensorBoard:** <https://huggingface.co/Toonies/>

- [11] <https://huggingface.co/Toonies/PredictFuture/tensorboard>
- [12] <https://huggingface.co/Toonies/Flowers/tensorboard>
- [13] <https://huggingface.co/Toonies/TenDishes/tensorboard>
- [14] <https://huggingface.co/Toonies/Money/tensorboard>
- [15] <https://huggingface.co/Toonies/FaceDetect/tensorboard>