

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----

**BÁO CÁO**

**BÀI TẬP LỚN**

*Đề tài: Phát hiện ảnh thật và ảnh tạo bằng AI sử dụng*

*các kỹ thuật học máy, học sâu*

**Giảng viên hướng dẫn: Vũ Minh Mạnh**

**Thành viên nhóm 8:**

1. **Nguyễn Thái Minh – B21DCCN090**
2. **Vũ Tuấn Nam - B21DCCN560**
3. **Dương Anh Đức – B21DCCN238**
4. **La Đức Hiệp – B21DCCN342**

**Hà Nội, 2024**

PHỤ LỤC

[CHƯƠNG 1: PHÂN CHIA CÔNG VIỆC 3](#_Toc165497377)

[1. Báo cáo đề tài 3](#_Toc165497378)

[CHƯƠNG 2: TỔNG QUAN VỀ DỰ ÁN 4](#_Toc165497379)

[1. Đặt vấn đề 4](#_Toc165497380)

[2. Mục tiêu 4](#_Toc165497381)

[3. Nội dung thực hiện 4](#_Toc165497382)

[CHƯƠNG 3: CƠ SỞ LÝ THUYẾT 6](#_Toc165497383)

[1. Giới thiệu về Machine Learning và Deep Learning 6](#_Toc165497384)

[1.1. Khái niệm về Machine Learning 6](#_Toc165497385)

[1.2. Khái niệm về Deep Learning 6](#_Toc165497386)

[2. Tìm hiểu về Tensorflow và Keras 6](#_Toc165497387)

[2.1. Tensorflow 6](#_Toc165497388)

[2.2. Keras 8](#_Toc165497389)

[3. Các phép toán ma trận 9](#_Toc165497390)

[3.1. Ảnh màu 9](#_Toc165497391)

[3.2. Tensor 9](#_Toc165497392)

[3.3. Ảnh xám 10](#_Toc165497393)

[3.4. Phép tính Convolution 11](#_Toc165497394)

[3.5. Padding 12](#_Toc165497395)

[3.6. Stride 13](#_Toc165497396)

[3.7. Ý nghĩa của phép tính Convolution 15](#_Toc165497397)

[4. Quá trình tạo lập dữ liệu đầu vào 16](#_Toc165497398)

[4.1. Tìm kiếm dữ liệu đầu vào 16](#_Toc165497399)

[4.2. Tiền xử lý hình ảnh 16](#_Toc165497400)

[4.3. Phân chia tập dữ liệu 18](#_Toc165497401)

[5. Thiết kế và xây dựng mô hình 19](#_Toc165497402)

[5.1. Convolutional Layer 19](#_Toc165497403)

[5.2. Pooling Layer 23](#_Toc165497404)

[5.3. Fully Connected Layer (Dense Layer) 24](#_Toc165497405)

[5.4. Xây dựng mô hình 25](#_Toc165497406)

[6. Huấn luyện và đánh giá mô hình. 26](#_Toc165497407)

[6.1. Huấn luyện 26](#_Toc165497408)

[6.2. Đánh giá mô hình 28](#_Toc165497409)

[CHƯƠNG 4: KẾT LUẬN 29](#_Toc165497410)

[1. Kết luận. 29](#_Toc165497411)

[2. Hướng phát triển 29](#_Toc165497412)

[TÀI LIỆU THAM KHẢO 30](#_Toc165497413)

# CHƯƠNG 1: PHÂN CHIA CÔNG VIỆC

## 1. Báo cáo đề tài

- Kính gửi: Thầy Vũ Minh Mạnh – Giảng viên bộ môn An toàn và bảo mật hệ thống thông tin.

- Nhóm 8 gồm bốn thành viên: Nguyễn Thái Minh, Vũ Tuấn Nam, Dương Anh Đức, La Đức Hiệp xin gửi đến thầy báo cáo đề tài: “Phát hiện ảnh thật và ảnh tạo bằng AI sử dụng các kỹ thuật học máy học sâu”.

|  |  |  |
| --- | --- | --- |
| Tên thành viên | Công việc | Phần trăm đóng góp |
| Dương Anh Đức | Thu thập, kiểm thử dataset là các ảnh | 25% |
| Vũ Tuấn Nam | Tìm hiểu, xây dựng kiến trúc AI | 25% |
| Nguyễn Thái Minh | Code, compile model và train AI | 25% |
| La Đức Hiệp | Tổng hợp, đánh giá và kiểm thử kết quả huấn luyện Model | 25% |

# CHƯƠNG 2: TỔNG QUAN VỀ DỰ ÁN

## 1. Đặt vấn đề

- Với sự phát triển mạnh mẽ của máy tính hiện nay, các phần mềm sử dụng thuật toán học máy học sâu đã trở nên phổ biến hơn hẳn. Từ việc sử dụng công cụ tìm kiếm trên Internet như Google, cho đến các chatbot phổ biến gần đây như là ChatGPT,... chúng đều tận dụng các AI tạo nên từ mạng lưới học máy, học sâu.

- Một trong những công dụng khác cũng vô cùng phổ hiển của AI hiện nay chính là tạo ra các hình ảnh chân thật. GAN (Generative Adversarial Networks) là một mạng lưới được tạo ra với công dụng như vậy. GAN đã vô cùng thành công trong lĩnh vực học máy, do nó có thể tạo ra những hình ảnh chất lượng vô cùng cao, gần như không thể phân biệt được với bản gốc. Tuy nhiên, GAN cũng có thể được sử dụng để tạo ra những khuôn mặt giả rất chân thật (đánh lừa được cả mắt người):



(Hình ảnh giả tạo bởi GAN)

- Sử dụng GAN, những kẻ gian có thể tạo ra hình ảnh người khác nhằm mục đích phá hoại uy tín, giả danh, và bóp méo sự thật,...

## 2. Mục tiêu

- Trong đề tài này, mục tiêu chính của nhóm chúng em là phát triển một trang web sử dụng Tensorflow để phân biệt hình ảnh khuôn mặt được tạo bởi AI với hình ảnh khuôn mặt có thực.

## 3. Nội dung thực hiện

- Nội dung 1: Giới thiệu về Machine Learning và Deep Learning

- Nội dung 2: Tìm hiểu về Tensorflow và Keras

- Nội dung 3: Tìm hiểu về các phép toán ma trận

- Nội dung 4: Tìm hiểu về quá trình tạo lập dữ liệu

- Nội dung 5: Thiết kế và xây dựng mô hình

- Nội dung 6: Huấn luyện và đánh giá mô hình đã thiết kế

- Nội dung 7: Đánh giá kết quả thực hiện

# CHƯƠNG 3: CƠ SỞ LÝ THUYẾT

## 1. Giới thiệu về Machine Learning và Deep Learning

## 1.1. Khái niệm về Machine Learning

- Machine learning (ML) hay học máy là một nhánh của trí tuệ nhân tạo (AI), tập trung sử dụng dữ liệu và các thuật toán để cho phép AI “bắt chước” cách con người học để cải thiện độ chính xác.

- Bài toán machine learning thường được chia làm hai loại là dự đoán (prediction) và phân loại (classification). Các bài toán dự đoán như dự đoán giá nhà, giá xe… Các bài toán phân loại như nhận diện chữ viết tay, nhận diện đồ vật…

## 1.2. Khái niệm về Deep Learning

- Deep Learning (DL) hay học sâu lại là một nhánh của học máy sử dụng mạng lưới Nơ-ron đa lớp để có thể giả lập các quyết định phức tạp hơn của con người.

- Theo định nghĩa, thì một mạng lưới Nơ-ron là mạng với 3 lớp hoặc nhiều hơn. Trên thực tế thì số mạng lưới còn nhiều hơn thế. Mạng lưới Nơ-ron được huấn luyện thông qua một lượng thông tin khổng lồ để có thể phát hiện và phân loại các hiện tượng, nhận diện khuôn mẫu và quan hệ, đánh giá các khả năng và đưa ra dự đoán, quyết định. Khi mạng lưới Nơ-ron được bổ sung thêm lớp mới sẽ cải thiện và tăng độ chính xác của kết quả đầu ra.

- Deep Learning chính là chìa khóa trong rất nhiều ứng dụng tự động hiện nay, giúp chúng ta phân tích thông tin mà không cần giám sát. Các chương trình như: Trợ lý ảo, điều khiển TV dùng giọng nói hay xe tự lái,... đều là sản phẩm nhờ việc sử dụng Deep Learning.

## 2. Tìm hiểu về Tensorflow và Keras

### 2.1. Tensorflow

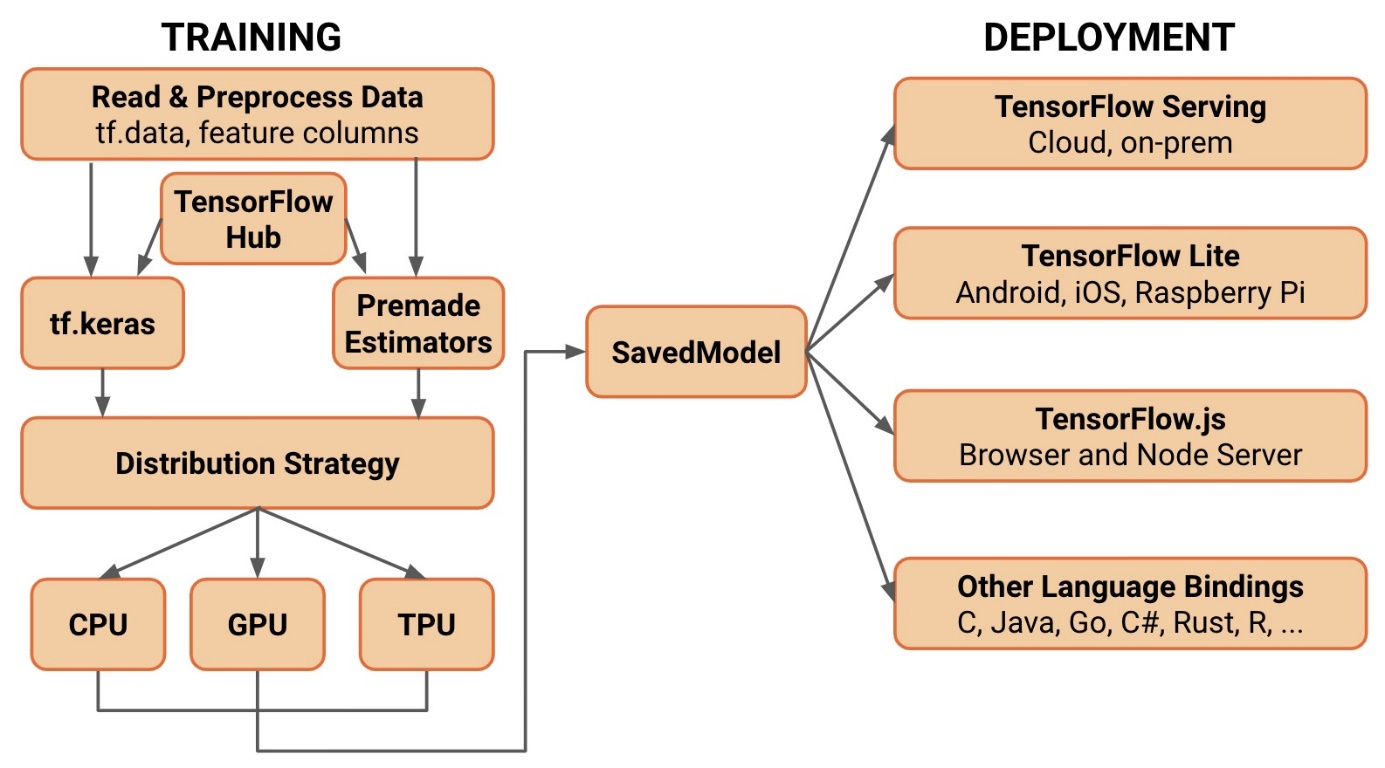
#### 2.1.1. Khái niệm

- Tensorflow là một thư viện mã nguồn mở phát triển bởi Google, được sử dụng để xây dựng các mô hình học máy và mạng nơ-ron. Tensorflow giúp giải quyết các bài toán nhanh chóng và đơn giản hơn thông qua việc tạo các mô hình tính toán trong việc học máy của máy tính.

#### 2.1.2. Nguyên lý hoạt động của Tensorflow

- Tensorflow là một thư viện tính toán số và biểu diễn dữ liệu bằng cấu trúc đồ thị (graph) để tạo và huấn luyện các mô hình học máy. Các đồ thị này bao gồm các nút (nodes) và các cạnh (edges) được sử dụng để biểu diễn các phép tính và dữ liệu tương ứng trong mô hình. Một cách tổng quát, quá trình huấn luyện mô hình học máy trong TensorFlow trải qua các bước sau:

* Xây dựng đồ thị tính toán: Người dùng xác định cấu trúc đồ thị tính toán bằng cách khai báo các biến (variables) và các phép tính (operations) trong mô hình.
* Định nghĩa hàm mất mát: Hàm mất mát (loss function) được định nghĩa để đo lường sự khác biệt giữa đầu ra dự đoán của mô hình và giá trị thực tế.
* Tối ưu hóa mô hình: Quá trình tối ưu hóa được sử dụng để tìm ra các giá trị tham số tối ưu nhằm giảm thiểu hàm mất mát.
* Huấn luyện mô hình: Dữ liệu huấn luyện được đưa vào mô hình để huấn luyện và cập nhật các giá trị tham số.
* Đánh giá mô hình: Dữ liệu kiểm tra được sử dụng để đánh giá hiệu suất của mô hình.
* Sử dụng mô hình: Mô hình đã huấn luyện được sử dụng để dự đoán và phân loại các dữ liệu mới.



#### 2.1.3. Các thuộc tính cơ bản của TensorFlow

- Về cơ bản, các thuộc tính của TensorFlow bao gồm:

* Tensor: Là đối tượng chính của TensorFlow, đại diện cho dữ liệu và kết quả tính toán. Tensor là một mảng đa chiều gồm các phần tử cùng kiểu dữ liệu.
* Operation: Là các hoạt động/phép toán được thực hiện trên các Tensor. Một số phép cơ bản của TensorFlow gồm có cộng, trừ, nhân và chia.
* Variable: Là các đối tượng được sử dụng để lưu trữ trạng thái thay đổi trong quá trình huấn luyện mô hình. Variable có thể được khởi tạo với giá trị cố định hoặc giá trị ngẫu nhiên và có thể được cập nhật trong quá trình huấn luyện.
* Graph: Là các biểu đồ đại diện cho các phép tính và quan hệ giữa chúng. Graph được sử dụng để mô tả cấu trúc của mô hình học máy và tạo ra các tính toán hiệu quả trên nhiều thiết bị tính toán.
* Session: Là một phiên làm việc của TensorFlow, chứa tất cả các biến và phép tính cần thiết để thực hiện tính toán.
* Placeholder: Là các đối tượng được sử dụng để đại diện cho các tensor được cấp cho mô hình trong quá trình huấn luyện hoặc kiểm tra. Placeholder được sử dụng để đảm bảo rằng dữ liệu đầu vào có thể được cung cấp cho mô hình.

### 2.2. Keras

- Keras là một thư viện mã nguồn mở Python được sử dụng rộng rãi trong lĩnh vực Deep Learning và mạng Nơ-ron. Nó được thiết kế để giúp các nhà phát triển xây dựng và thử nghiệm các mô hình học sâu một cách dễ dàng và linh hoạt.

- Mục tiêu chính của Keras là cung cấp một API đơn giản, trực quan dễ sử dụng, giúp người dùng tập trung vào việc xây dựng mô hình mà không cần lo lắng về chi tiết ký thuật phức tạp.

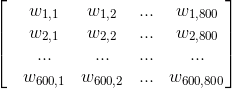
- Một trong những điểm mạnh của Keras là khả năng tích hợp với các framework học sâu khác như TensorFlow, Theano, PlaidML, MXNet, CNTK,... , cho phép người dùng tận dụng sức mạnh của cả hai thế giới. Đây chính là điều giúp Keras trở thành một lựa chọn hàng đầu cho các dự án học sâu từ phân loại ảnh, dự đoán chuỗi thời gian, đến các ứng dụng trong lĩnh vực ngôn ngữ tự nhiên và nhiều lĩnh vực khác.

- Trong rất nhiều framework khác nhau ở trên, TensorFlow đã tích hợp Keras làm API chính thức.

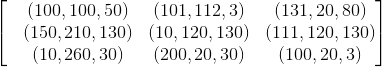
## 3. Các phép toán ma trận

### 3.1. Ảnh màu

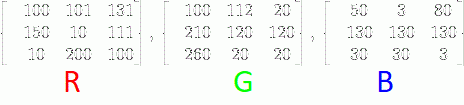
- Giả sử ta có một tấm ảnh kích cỡ 800 x 600 pixel, ta có thể biểu diễn ảnh đó dưới dạng ma trận cỡ 600 x 800, với mỗi phần tử là một pixel.



- Tuy nhiên để biểu diễn một màu ta sẽ cần 3 thông số r, g, b, nên gọi *wij*​=(*rij*​,*gij*​,*bij*​) để biểu diễn dưới dạng ma trận, ta sẽ có kết quả như sau:



- Để tiện lưu trữ và xử lý ta sẽ không thể lưu dưới dạng như trên được mà phải tách mỗi giá trị của pixel ra thành một ma trận riêng:

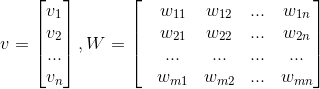


(3 ma trận cùng kích thước, mỗi ma trận lưu giá trị từng màu Red, Green, Blue)

### 3.2. Tensor

- Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột.

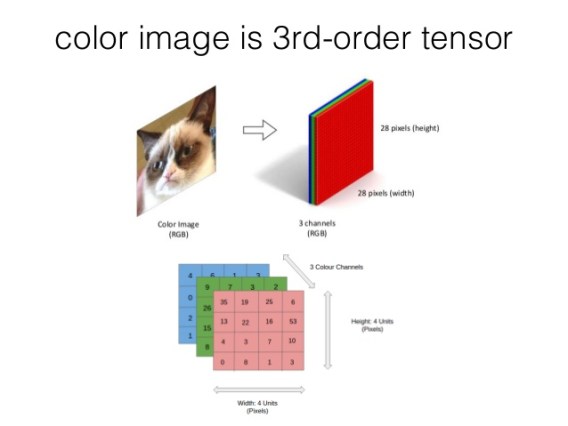
- Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước là số hàng \* số cột.



(Vector v kích thước n và ma trận W kích thước m\*n)

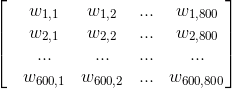
- Khi dữ liệu nhiều hơn 2 chiều thì sẽ được gọi là tensor. Tensor có thể coi là sự kết hợp của các ma trận cùng kích thước, xếp k ma trận kích thước m\*n chồng lên nhau sẽ được tensor kích thước m\*n\*k.

- Ví du biểu diễn ảnh màu kích thước 28\*28, ta sẽ biểu diễn dưới dạng tensor 28\*28\*3:

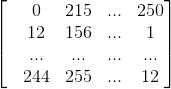


### 3.3. Ảnh xám

- Tương tự ảnh màu, ảnh xám cũng có kích thước 800 pixel \* 600 pixel, có thể biểu diễn dưới dạng một [ma trận](https://nttuan8.com/bai-1-linear-regression-va-gradient-descent/#Ma_tran) kích thước 600 \* 800 (vì định nghĩa ma trận là số hàng nhân số cột).



- Tuy nhiên mỗi pixel trong ảnh xám chỉ cần biểu diễn bằng một giá trị nguyên trong khoảng từ [0,255] thay vì (r,g,b) như trong ảnh màu. Do đó khi biểu diễn ảnh xám trong máy tính chỉ cần một ma trận là đủ.

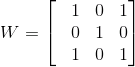


Biểu diễn ảnh xám

- Giá trị 0 là màu đen, 255 là màu trắng và giá trị pixel càng gần 0 thì càng tối và càng gần 255 thì càng sáng.

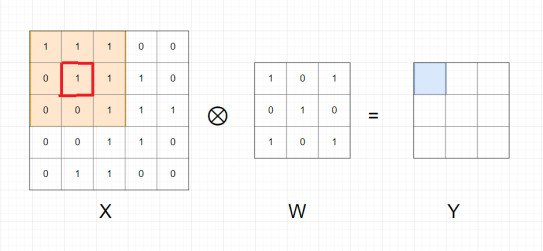
### 3.4. Phép tính Convolution

- Ta định nghĩa **kernel**là một ma trận vuông kích thước k\*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9,… Ví dụ kernel kích thước 3\*3



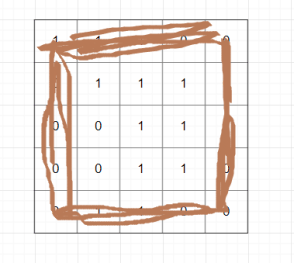
- Kí hiệu phép tính convolution (⊗), kí hiệu Y =*X*⊗*W*

- Với mỗi phần tử *xij*​ trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử *xij*​ làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính [element-wise](https://nttuan8.com/bai-1-linear-regression-va-gradient-descent/#Element-wise_multiplication_matrix) của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.

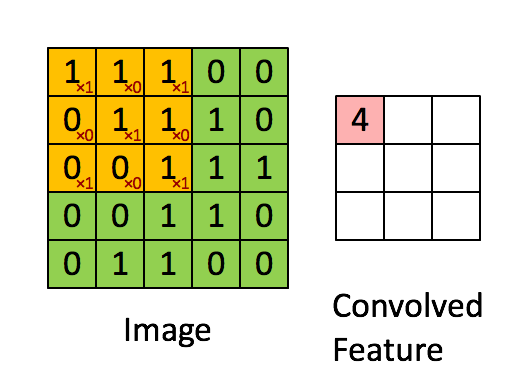


Ví dụ khi tính tại *x*22​ (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có *x*22​ làm trung tâm có màu nền da cam như trong hình. Sau đó tính 𝑦11=𝑠𝑢𝑚(𝐴⊗𝑊)=𝑥11∗𝑤11+𝑥12∗𝑤12+𝑥13∗𝑤13+𝑥21∗𝑤21+𝑥22∗𝑤22+𝑥23∗𝑤23+𝑥31∗𝑤31+𝑥32∗𝑤32+𝑥33∗𝑤33. Và làm tương tự với các phần tử còn lại trong ma trận.

- Thế thì sẽ xử lý thế nào với phần tử ở viền ngoài như *x*11​? Bình thường khi tính thì sẽ bỏ qua các phần tử ở viền ngoài, vì không tìm được ma trận A ở trong X.



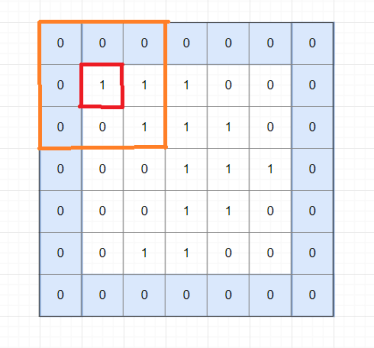
Nên ta có thể thấy ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y là (m-k+1) \* (n-k+1).



Các bước thực hiện phép tính convolution cho ma trận X với kernel K ở trên

### 3.5. Padding

Như ở trên thì mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên giờ ta muốn ma trận Y thu được có kích thước bằng ma trận X => Ta sẽ thêm giá trị 0 ở viền ngoài ma trận X.



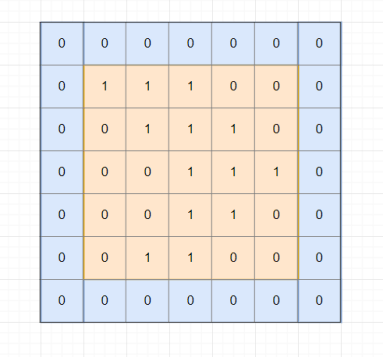
Ma trận X khi thêm viền 0 bên ngoài

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phần tử *x*11​ , và ma trận Y thu được sẽ bằng kích thước ma trận X ban đầu.

Phép tính này gọi là convolution với **padding=1**. Padding=k nghĩa là thêm k vector 0 vào mỗi phía của ma trận.

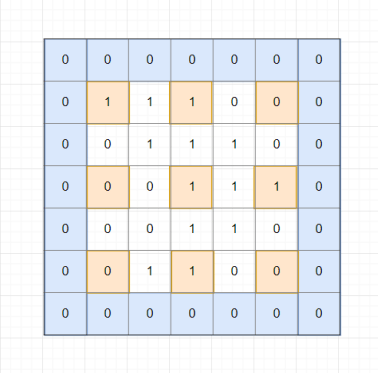
### 3.6. Stride

Như ở trên ta thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.



stride=1, padding=1

Tuy nhiên nếu **stride=k** (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử 𝑥(1+𝑖∗𝑘,1+𝑗∗𝑘)*.*Ví dụ k = 2.



padding=1, stride=2

Hiểu đơn giản là bắt đầu từ vị trí *x*11​ sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X.

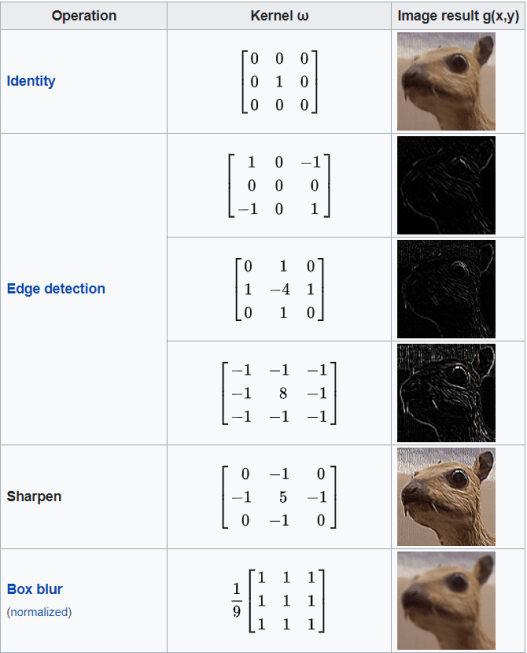
Kích thước của ma trận Y là 3\*3 đã giảm đi đáng kể so với ma trận X.

Công thức tổng quát cho phép tính convolution của ma trận X kích thước m\*n với kernel kích thước k\*k, stride = s, padding = p ra ma trận Y kích thước ((𝑚−𝑘+2𝑝)/𝑠+1)∗((𝑛−𝑘+2𝑝)/𝑠+1).

Stride thường dùng để giảm kích thước của ma trận sau phép tính convolution.

### 3.7. Ý nghĩa của phép tính Convolution

- Sử dụng phép Convolution, ta có thể thực hiện làm mờ, nét ảnh , xác định các đường,... Mỗi kernel khác nhau thì phép tính convolution sẽ có ý nghĩa khác nhau. Ví dụ:



## 4. Quá trình tạo lập dữ liệu đầu vào

### 4.1. Tìm kiếm dữ liệu đầu vào

- Chúng em sử dụng trang [kaggle.com](https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces) để tìm một bộ dữ liệu phù hợp cho việc nhận diện khuôn mặt. Kết quả tìm được một bộ dữ liệu bao gồm 70.000 ảnh mặt thật và 70.000 ảnh khuôn mặt giả (tạo bởi StyleGAN). Tất cả những hình ảnh đều có kích cỡ 256x256px.

### 4.2. Tiền xử lý hình ảnh

- Để có thể tạo ra mạng lưới Nơ-ron hiệu quả, chúng em cần cân nhắc chuẩn hóa dạng thông tin đầu vào.

- Chúng em tạo một hàm với tham số là đường dẫn tới ảnh, đọc ảnh từ folder, và trả lại hình ảnh đó sau khi đã áp dụng các bước tiền xử lý.

def read\_and\_preprocess(img\_path):

    img = cv2.imread(img\_path, cv2.IMREAD\_COLOR)

    img = cv2.resize(img, (IMAGE\_SIZE, IMAGE\_SIZE))

    img = img.astype('float32') / 255.0  # Normalize to [0, 1]

    return img

- Hàm imread() sẽ tải hình ảnh đọc được từ đường dẫn, sau đó ảnh được chỉnh kích cỡ về 128x128 rồi ảnh sẽ được “normalize”.

- Normalize là quá trình quan trọng giúp đảm bảo dữ liệu hình ảnh đầu vào (là các pixel) có sự phân bố giá trị màu đồng đều hơn.

- Các ảnh sau khi được chuẩn hóa sẽ gồm ma trận có giá trị các pixel nằm trong khoảng từ 0 – 1.



(Một số hình ảnh gốc chưa chuẩn hóa)



(Hình ảnh sau khi đã chuẩn hóa)

### 4.3. Phân chia tập dữ liệu

- Phân chia tập dữ liệu cũng là một bước vô cùng quan trọng để giảm thiểu “quá khớp” (khi model học cả phần nhiễu trong dữ liệu thay vì chỉ học mỗi khuôn mẫu). Một model quá khớp sẽ chỉ hoạt động tốt trên data được học, nhưng không thể hoạt động được trên data dùng để test.

- Chúng em chia tổng 140.000 hình ảnh thành 100.000 ảnh huấn luyện cho model, 20.000 ảnh để test, 20.000 ảnh để kiểm chứng. Mỗi phần đều đảm bảo các điều kiện sau:

* Mỗi phần đều có tỉ lệ 50/50 giữa ảnh thật và ảnh giả.
* Các mẫu ảnh được đảo ngẫu nhiên và sử dụng chung một random state để đảm bảo mỗi lần phân tách luôn tạo ra cùng một kết quả.

for folder in os.scandir(image\_path):

    for entry in os.scandir(image\_path + folder.name):

        X.append(read\_and\_preprocess(image\_path + folder.name + '/' + entry.name))

        y.append(0 if folder.name[0] == 'r' else 1)  # Real: 0, Fake: 1

X = np.array(X)

y = np.array(y)

#Split data into training, validation, and testing

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=123)

X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_val, y\_val, test\_size=0.5, stratify=y\_val, random\_state=123)

## 5. Thiết kế và xây dựng mô hình

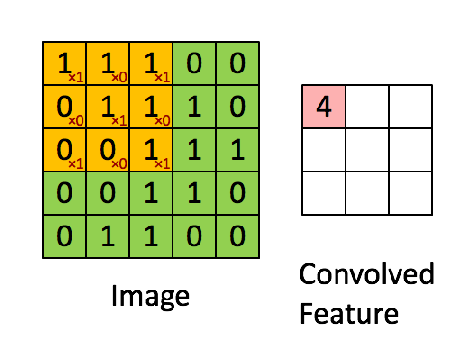
### 5.1. Convolutional Layer

- Trong một bức ảnh, thường các pixel ở gần nhau sẽ có “liên kết” với nhau hơn là những pixel ở xa (Như các pixel biểu diễn một đường trên mặt người). Để tìm các đường trong ảnh, ta áp dụng Sobel Kernel trên mỗi vùng kích thước 3\*3, hay để tìm các ảnh ta áp dụng Sharpen Kernel cũng trên vùng có kích thước 3\*3.

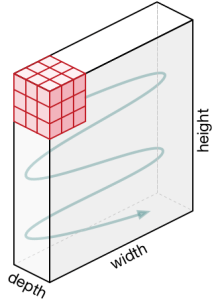
- Trong khi tính 1 Convolution trong ảnh, chúng ta chỉ dùng 1 kernel cho toàn bộ tất cả các bức ảnh, như vậy sẽ đảm bảo các pixel của các ảnh chia sẻ hệ số với nhau.

-> Áp dụng các phép tính Convolution liên tục, ta có thể tìm ra được các đặc trưng của ảnh.

- Như mục 3.4, ta có phép tính Convolution thực hiện trên ảnh xám với biểu diễn ảnh dạng ma trận:

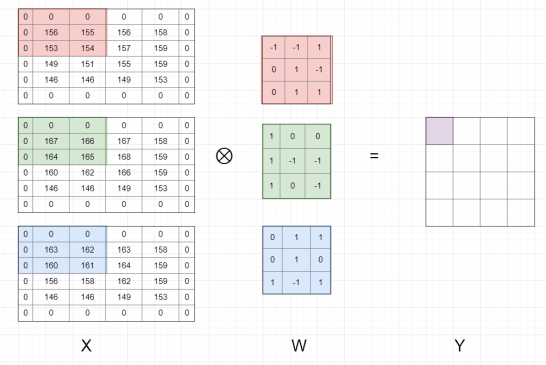


- Tuy nhiên ảnh màu có tới 3 channel red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều, ta cũng sẽ định nghĩa kernel là tensor 3 chiều kích thước k\*k\*3.



Phép tính convolution trên ảnh màu với k=3.

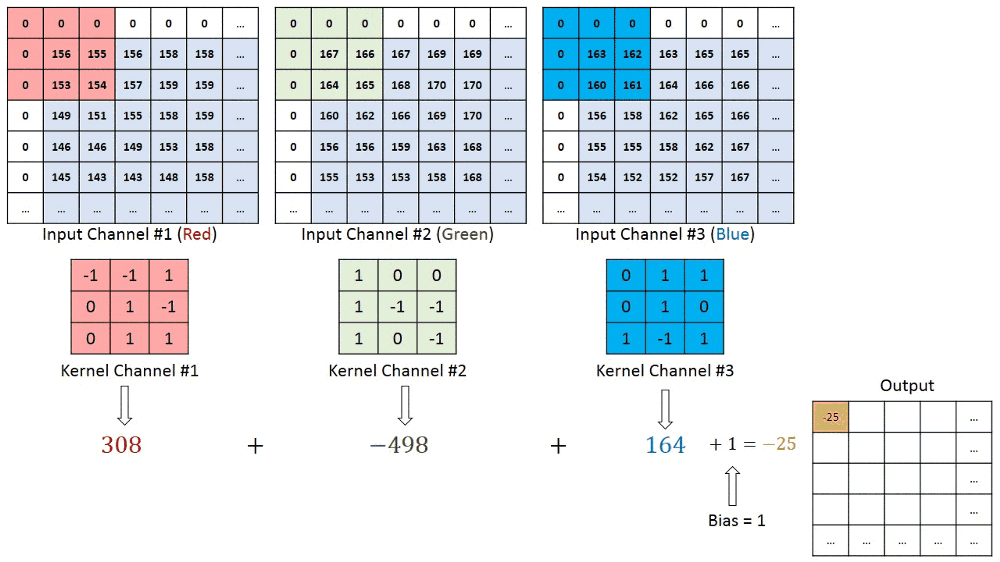
Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám.



Tensor X, W 3 chiều được viết dưới dạng 3 ma trận.

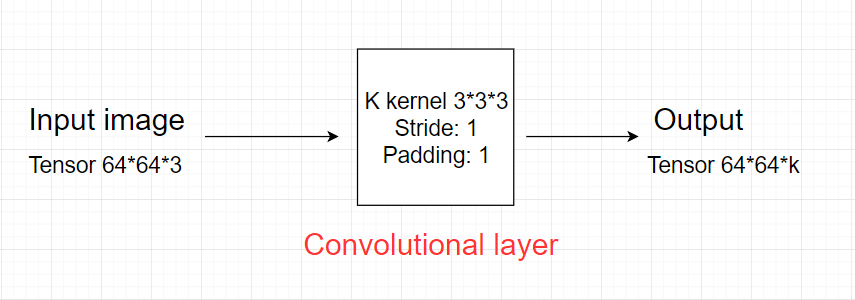
Khi biểu diễn ma trận ta cần 2 chỉ số hàng và cột: i và j, thì khi biểu diễn ở dạng tensor 3 chiều cần thêm chỉ số độ sâu k. Nên chỉ số mỗi phần tử trong tensor là *xijk*​. Ta khi đó có thể tính giá trị của y11:

𝑦11=𝑏+(𝑥111∗𝑤111+𝑥121∗𝑤121+𝑥131∗𝑤131+𝑥211∗𝑤211+𝑥221∗𝑤221+𝑥231∗𝑤231+𝑥311∗𝑤311+𝑥321∗𝑤321+𝑥331∗𝑤331)+(𝑥112∗𝑤112+𝑥122∗𝑤122+𝑥132∗𝑤132+𝑥212∗𝑤212+𝑥222∗𝑤222+𝑥232∗𝑤232+𝑥312∗𝑤312+𝑥322∗𝑤322+𝑥332∗𝑤332)+(𝑥113∗𝑤113+𝑥123∗𝑤123+𝑥133∗𝑤133+𝑥213∗𝑤213+𝑥223∗𝑤223+𝑥233∗𝑤233+𝑥313∗𝑤313+𝑥323∗𝑤323+𝑥333∗𝑤333) =−25

Thực hiện phép tính convolution trên ảnh màu

- Ta có thể thấy output Y của phép tính convolution trên ảnh là một ma trận, và có một hệ số bias được cộng vào sau bước tính tổng các phần tử của phép tính element-wise.

- Với mỗi kernel khác nhau ta sẽ tìm được đặc trưng khác nhau của ảnh, nên trong mỗi convolutional layer ta sẽ dùng nhiều kernel để học được nhiều thuộc tính của ảnh. Vì mỗi kernel output ra là một ma trận nên k kernel sẽ cho ra k ma trận output. Ta kết hợp k ma trận output lại thành 1 tensor 3 chiều có chiều sâu k.



Convolution layer đầu tiên

- k ma trận output (Tensor output) của Convolutional layer đầu tiên sẽ được sử dụng cho Convolutional layer kế tiếp.

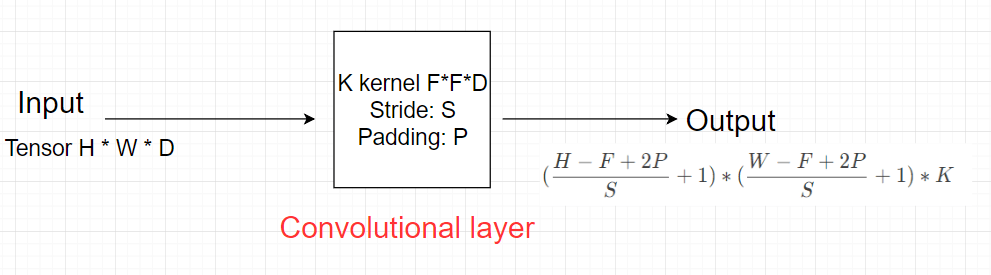
\* Convolutional layer tổng quát:

- Giả sử input của 1 convolutional layer tổng quát là tensor kích thước H \* W \* D.

- Kernel có kích thước F \* F \* D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.

- Convolutional layer áp dụng K kernel.

-> Output của layer sẽ là tensor 3 chiều có kích thước: ((*H*−*F*+2*P*​)/S+1)∗((*W*−*F*+2P)/S​+1)∗*K*

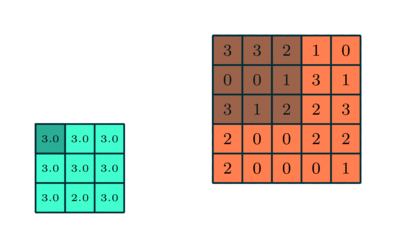


- Tổng số parameter của layer: Mỗi kernel có kích thước F\*F\*D và có 1 hệ số bias, nên tổng parameter của 1 kernel là F\*F\*D + 1. Mà convolutional layer áp dụng K kernel => Tổng số parameter trong layer này là K \* (F\*F\*D + 1).

### 5.2. Pooling Layer

- Pooling layer được dùng giữa các Convolutional layer để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm làm cho việc tính toán trở nên dễ dàng hơn trong model.

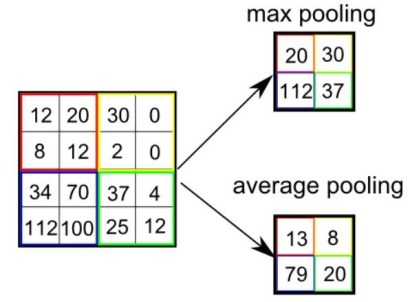
- Gọi pooling size kích thước K\*K. Input của pooling layer có kích thước H\*W\*D, ta tách ra làm D ma trận kích thước H\*W. Với mỗi ma trận, trên vùng kích thước K\*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.



Max pooling layer với size=(3,3), stride=1, padding=0

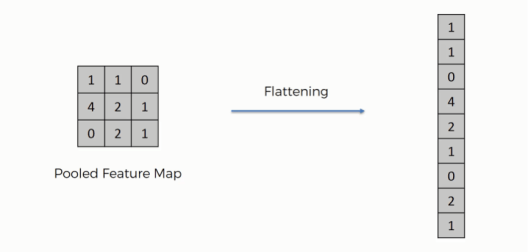
- Có thể thấy kích thước của ma trận đã giảm đi.

- Có hai loại Pooling layer phổ biến là: Max Pooling và Average Pooling. Ở bài thực hành này chúng em sử dụng Max Pooling.



### 5.3. Fully Connected Layer (Dense Layer)

- Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,…), khi đó tensor của output của layer cuối cùng, kích thước H\*W\*D, sẽ được chuyển về 1 vector kích thước (H\*W\*D).



- Sau đó ta sẽ dùng các fully connected layer để kết hợp các đặc điểm của ảnh để tạo ra output của model.

### 5.4. Xây dựng mô hình

#### 5.4.1. Khái quát mô hình

- Sử dụng thư viện TensorFlow, chúng em có thể dễ dàng tạo một model dạng tuần tự với cấu trúc như sau:

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(IMAGE\_SIZE, IMAGE\_SIZE, CHANNELS)),

    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu', padding='same'),

    MaxPooling2D((2, 2)),

    Dropout(0.3),

    Conv2D(64, (3, 3), activation='relu', padding='same'),

    Conv2D(128, (3, 3), activation='relu', padding='same'),

    MaxPooling2D((2, 2)),

    Dropout(0.4),

    Conv2D(128, (3, 3), activation='relu', padding='same'),

    MaxPooling2D((2, 2)),

    Dropout(0.5),

    Flatten(),

    Dense(512, activation='relu'),

    Dropout(0.5),

    Dense(256, activation='relu'),

    Dropout(0.3),

    Dense(128, activation='relu'),

    Dense(2, activation='softmax')  # Output layer with 2 units for 2 classes

])

- Model bao gồm 5 Convolution layer với kernel cỡ 3\*3.

* Lớp Convolutional đầu tiên có 32 filter (32 kernel) với kích cỡ 3\*3.
* Sau mỗi lớp Convolutional, số kernel sẽ tăng gấp đôi.
* Để tránh hiện tượng quá khớp và giảm chi phí tính toán, lớp Max Pooling sẽ được đặt ở giữa các lớp Convolutional.
* Kết quả từ các lớp Convolutional sau đó sẽ được làm phẳng (chuyển về vector) rồi truyền đến lớp Dense.
* Lớp Dense đầu tiên có 512 node, và sẽ chia đôi với mỗi lớp Dense tiếp theo.
* Lớp Dropout được xen vào giữa để giảm thiểu hiện tượng quá khớp.
* Hàm kích hoạt ReLU được sử dụng trong mọi layer, ngoại trừ layer output với 2 node và hàm activation là softmax.

#### 5.4.2. Các tham số.

- Convo2D: Lớp Convolutional.

* 32: Số filter (Số kernel) của lớp.
* (3,3): Kích thước của Kernel.
* activation=’relu’: Sử dụng hàm ReLU áp dụng cho các giá trị đầu ra của Layer, hàm ReLU chỉ đơn giản là đưa các giá trị nhỏ hơn 0 về 0, còn giá trị dương giữ nguyên.
* input\_shape=(IMAGE\_SIZE, IMAGE\_SIZE, CHANNELS): Là dạng data được truyền vào lớp Convolution, với IMAGE\_SIZE là kích cỡ ảnh và là kích cỡ chiều dài và rộng của tensor đầu vào. CHANNELS là số kênh màu, và đại diện cho độ sâu của tensor.
* padding=’same’: Sử dụng padding giống như những layer trước, mặc định ở đây là 1.

- MaxPooling2D: Lớp Max Pooling.

* (2,2): Kích cỡ của ma trận.

- Dropout:

* 0.3: Tỉ lệ đặt một giá trị đầu vào thành 0 trong mỗi bước thực hiện training.

- Flatten: Chỉ là hàm biến tensor đầu ra của lớp Convolutional thành một vector.

- Dense:

* 512: Số node
* activation='softmax': Sử dụng hàm softmax.

## 6. Huấn luyện và đánh giá mô hình.

### 6.1. Huấn luyện

- Trước khi thực hiện huấn luyện cho mô hình, ta cần phải chú ý tới 3 yếu tố.

* Hàm mất mát (Loss function): Hàm mất mát đảm nhiệm việc đánh giá độ chính xác của kết quả. Hàm mất mát có giá trị càng cao thì càng thiếu chính xác và ngược lại. Chúng ta sử dụng hàm loss “sparse\_categorical\_crossentropy” vì ta đang cần phân loại các ma trận với đơn vị từ 0 – 1.
* The optimizer (Tối ưu): Cập nhật weight cho các kernel trong quá trình training để làm giảm thiểu hàm mất mát. Chúng ta sẽ sử dụng Adam.
* Evaluation (Đánh giá): Đánh giá quá trình training và validation (xác minh). Vì dataset các ảnh của chúng ta đã được sắp xếp chuẩn, ta sẽ đặt tiêu chí là độ chính xác (accuracy).

model.compile(loss='sparse\_categorical\_crossentropy', optimizer=Adam(learning\_rate=0.001), metrics=['accuracy'])

- Những tham số khác như tốc độ học, kích cỡ, số vòng lặp đều có thể được đặt trong quá trình compile model.

- Để có thể cải thiện tốc độ học của model, chúng ta có thể sử dụng 2 hàm:

* EarlyStopping: Giúp ngăn chặn hiện tượng quá khớp và cải thiện thời gian học thông qua việc giám sát hiệu năng của model trong quá trình xác thực (validation). Nó sẽ lập tức ngừng quá trình huấn luyện khi kết quả học của model trong quá trình xác thực bắt đầu giảm đi (Model đã đạt tới kết quả tốt nhất).

EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=10)

Hàm sẽ giám sát giá trị mất mát thấp nhất trong quá trình xác thực với patience = 10. Nghĩa là nếu như giá trị mất mát không giảm đi sau 10 vòng lặp (epoch) thì quá trình huấn luyện sẽ ngừng lại.

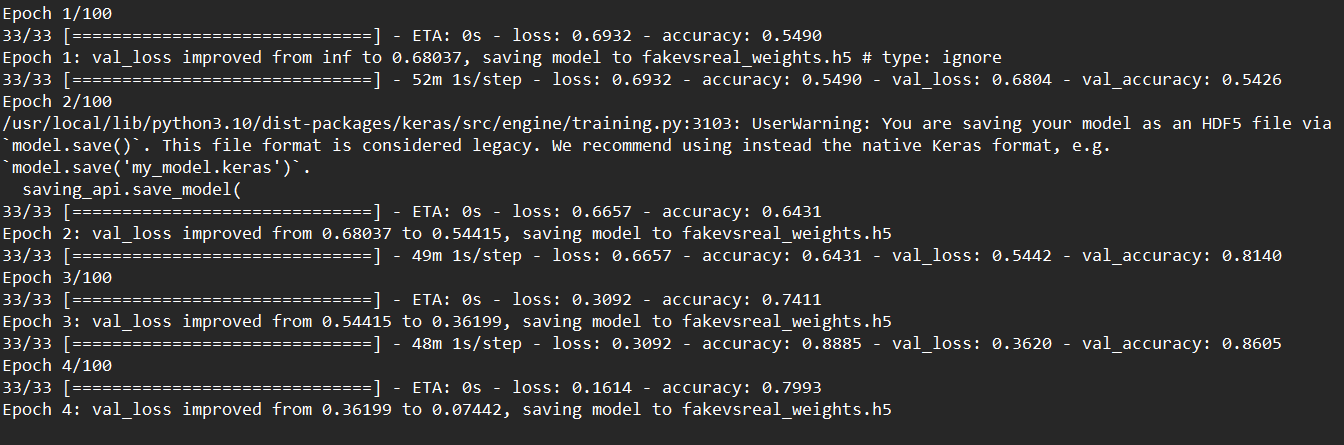
* ModelCheckpoint: Hàm này sẽ lưu lại tham số weight của các kernel trong quá trình huấn luyện. Quá trình này sẽ giúp chúng ta tiếp tục train model trong trường hợp gặp lỗi không mong muốn.

ModelCheckpoint(filepath="fakevsreal\_weights.h5", verbose=1, save\_best\_only=True)

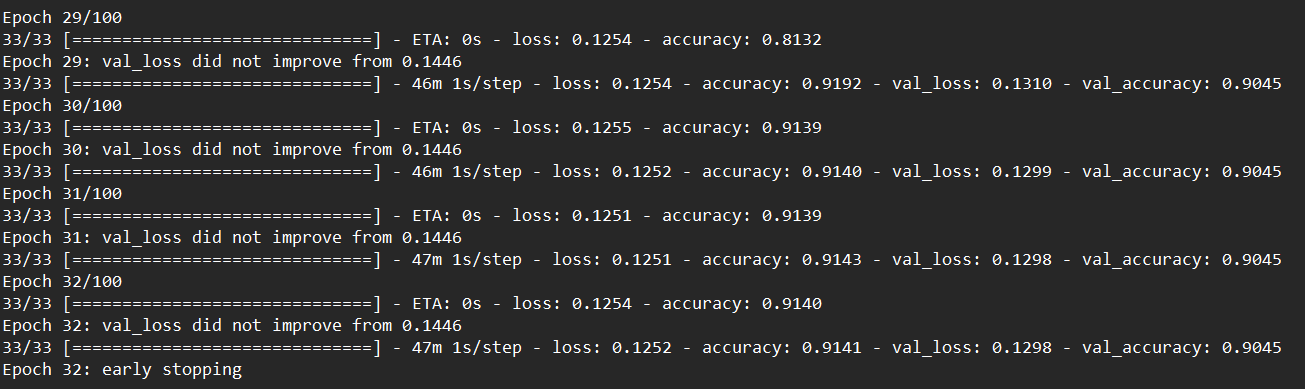
- Cuối cùng, ta chỉ việc truyền các tham số cần thiết và tiến hành train model.

model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_val, y\_val), batch\_size=BATCH\_SIZE, shuffle=True, callbacks=[earlystopping, checkpointer])

### 6.2. Đánh giá mô hình

****

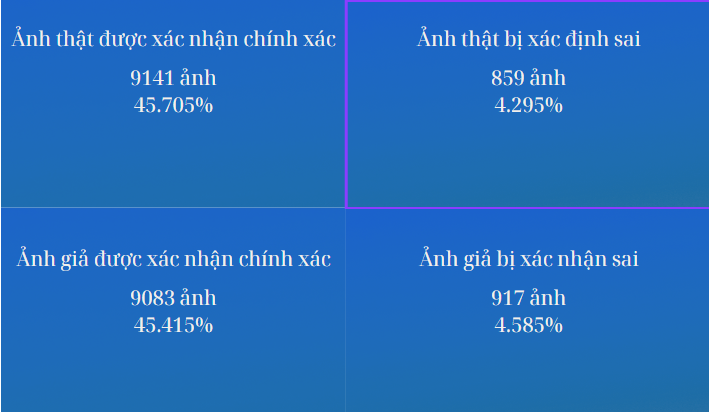
- Có thể thấy trải qua mỗi Epoch, giá trị loss giảm dần và độ chính xác tăng dần.



- Đến epoch (vòng lặp) thứ 32, thì giá trị độ chính xác (val\_accuracy) không còn cải thiện nữa, nên hàm EarlyStopping sẽ chạy và ngừng quá trình huấn luyện.

- Sau EarlyStopping, model dừng lại với độ chính xác vào ngưỡng 0.9141 (91.41% chính xác)

- Dưới đây là kết quả thử với 20000 hình ảnh, với 10000 ảnh thật và 10000 ảnh giả.



- Kết quả thử cho thấy tổng số % ảnh bị xác định thật giả sai chiếm 8.88%, gần tương đương với kết quả độ chính xác trong khi train.

# CHƯƠNG 4: KẾT LUẬN

## 1. Kết luận.

- Sau khi tìm hiểu về đề tài “Phát hiện ảnh thật và ảnh tạo bằng AI sử dụng

các kỹ thuật học máy, học sâu”, nhóm đã thực hiện được theo yêu cầu cơ bản ban đầu. Và đã thu được một số kết quả như sau:

* Nắm được các khái niệm cơ bản về AI, machine learning và deep learning.
* Học được các phép toán chuẩn hóa hình ảnh như phép nhân Convolution.
* Sử dụng các phép toán để dựng nên kiến trúc AI cơ bản.
* Thu thập dữ liệu thông tin về hình ảnh các khuôn mặt.
* Sử dụng dữ liệu hình ảnh thu thập được vào việc huấn luyện AI.
* Học được cách sử dụng thư viện TensorFlow cũng như API của nó, Keras.

- AI có thể nói là đã đạt được kết quả mong muốn, song vẫn còn một số hạn chế như nhiễu từ ảnh nền có thể ảnh hưởng tới kết quả xác định khuôn mặt, hay “khuôn mặt thứ 2” trong những ảnh giả tạo bởi AI cũng có thể gây nhiễu,...

2. Hướng phát triển**.**

- Việc nghiên cứu sâu hơn về AI phát hiện ảnh giả tạo bởi AI cần rất nhiều thời gian. Nhóm chúng em nhận thấy có thể phát triển thêm như sau:

* Mở rộng phạm vi xác minh của AI từ chỉ khuôn mặt ra các lĩnh vực khác như chữ viết tạo bởi AI.
* Huấn luyện AI linh hoạt hơn thông qua nhiều dữ liệu hơn, rồi từ đó có thể áp dụng vào việc phát hiện các video giả mạo như video tạo bởi công nghệ deepfake.

# TÀI LIỆU THAM KHẢO

1. Tiền xử lý hình ảnh.

<https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258>

2. TensorFlow

<https://www.tensorflow.org/learn>

3. Keras API

<https://www.tensorflow.org/guide/keras>

4. GAN

<https://www.analyticsvidhya.com/blog/2021/03/why-are-generative-adversarial-networksgans-so-famous-and-how-will-gans-be-in-the-future/#:~:text=GANs%20are%20usually%20trained%20to,digit%20images%20that%20look%20like>

5. CNN (Convolutional Neural Network)

<https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>

6. Convolutional Theorem

<https://en.wikipedia.org/wiki/Convolution_theorem>