UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

PROJECT03 - IMAGE CLASSIFICATION

COURSE: ARTIFICIAL INTELLIGENCE

Instructors: Mr. Châu Thành Đức

Mrs. Phan Thị Phương Uyên

Mr. Ngô Đình Hy

Team: 19127078 - Nguyễn Đỗ Thanh Trúc

19127097 - Nguyễn Ngọc Phương Anh

19127142 - Trần Thái Đức Hiếu

Ho Chi Minh City, 08/21/2021

Các khái niệm mở đầu:

Các loại data set:

1. Training set

- Training set là tập dữ liệu có input và output (trong bài toán Image Classification, input sẽ là những bức ảnh, output là nhãn ứng với input) dùng để huấn luyện (train) model.

2. Validation set

- Validation set thường là tập dữ liệu con được trích ra từ training set dùng để kiểm tra độ chính xác của model trong quá trình huấn luyện, từ đó tối ưu hóa model.

3. Test set

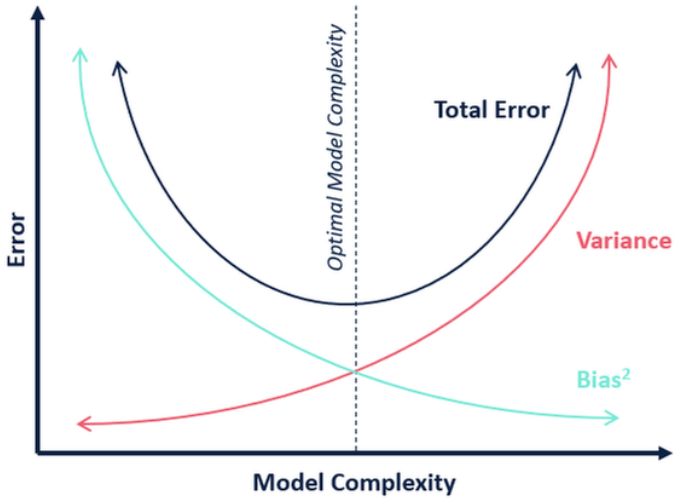
- Test set là tập dữ liệu chỉ bao gồm input, dùng để kiểm tra độ chính xác của model sau quá trình huấn luyện. (Không dùng test set trong quá trình huấn luyện model)

Vấn đề về Underfitting và Overfitting:

1. Cách để đánh giá model:

- Sử dụng hàm dự đoán lỗi **Total error**, bao gồm:
    - **Bias**: Độ lệch đặc trưng cho sự chênh lệch giữa giá trị trung bình mà model dự đoán so với giá trị đúng.
    - **Variance**: Phương sai đặc trưng cho độ phân tán của dữ liệu so với giá trị trung bình, thể hiện mức độ nhạy cảm với sự biến động nhỏ trong training set.
    - **Irreducible error**: Lỗi không thể loại bỏ cho dù lựa chọn các thuật toán khác nhau (nhiễu (noise), ngẫu nhiên, phân loại sai).

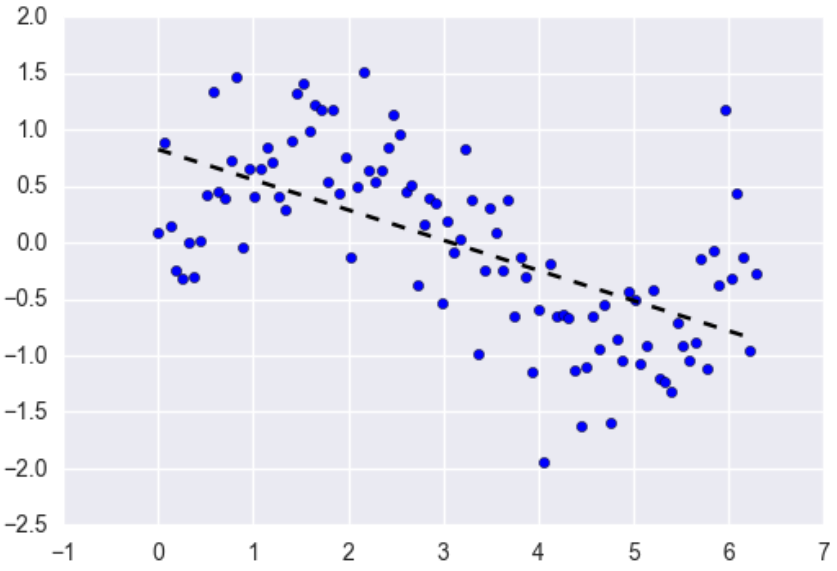
$$\textbf{Total error} = \textbf{Bias}^2 + \textbf{Variance} + \textbf{Irreducible error}$$



Source: Analytics Vidhya

2. Underfitting

- Underfitting là hiện tượng khi model dự đoán không chính xác trên cả training set và test set, xảy ra khi model không khớp với training set.



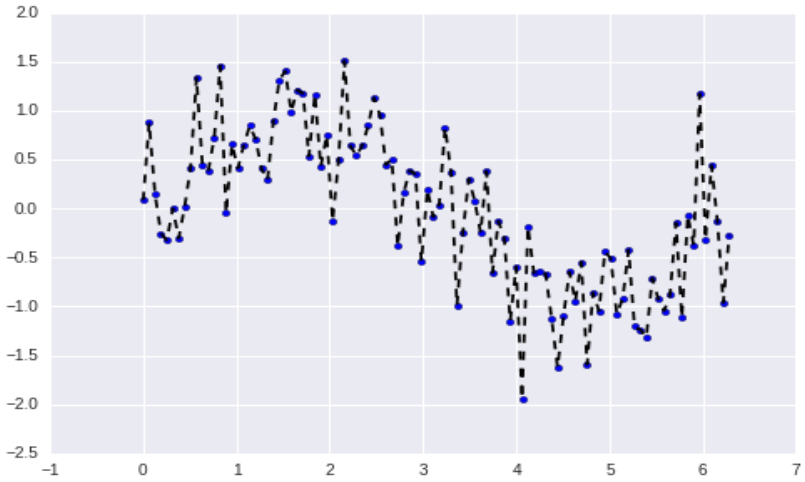
Source: Elite Data Science

- **Nguyên nhân:** Do model quá đơn giản còn dữ liệu thì quá phức tạp.
- **Cách nhận biết:** Khi mô hình cho kết quả độ lệch lớn (high bias) nhưng phương sai nhỏ (low variance) => Model không quan tâm nhiều tới training set, khiến cho model trở nên đơn giản quá.

- **Cách giải quyết:** Cải tiến thuật toán (tăng độ phức tạp: thêm layer, node, …), bổ sung thêm dữ liệu đầu vào.

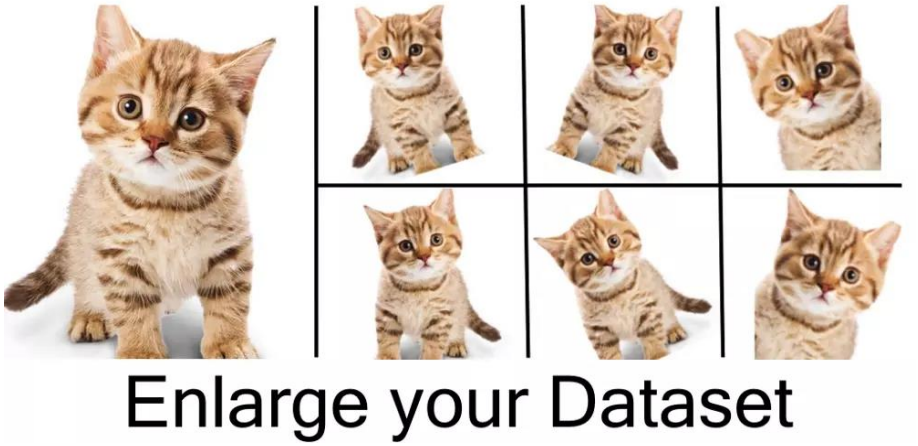
3. Overfitting

- Overfitting là hiện tượng khi model dự đoán quá chính xác với training set nhưng lại không hiệu quả với test set, xảy ra khi model quá khớp với training set.



Source: Elite Data Science

- **Nguyên nhân:** Do training set có quá nhiều nhiễu (noise), dữ liệu bất thường, hoặc model quá phức tạp (khi training set nhỏ nhưng model lại có độ phức tạp cao).
- **Cách nhận biết:** Khi mô hình cho kết quả độ lệch nhỏ (low bias) nhưng phương sai lớn (high variance) => Model tập trung quá nhiều vào training set dẫn đến không thể hiện tính tổng quát trên dữ liệu mới (tính tổng quát là tính mô tả được nhiều dữ liệu, trong cả tập training và test).
- **Các giải quyết:**
    1. **Tăng kích cỡ training set**
- **Thu thập thêm nhiều dữ liệu mới** (phải đảm bảo dữ liệu "sạch" (không nhiễu) và liên quan tới bài toán đang giải quyết)
  => Tốn kém (vì không có nhiều data mới để train)
- **Tăng cường dữ liệu(Data augmentation)**: Làm đa dạng training set bằng cách biến đổi data có sẵn (crop, flip, scale, …).
  => Thông dụng hơn do ít tốn kém so với việc kiếm thêm dữ liệu mới.



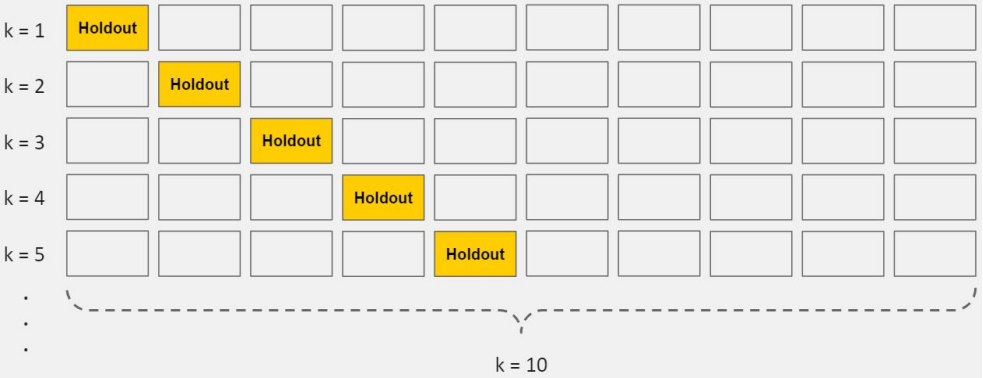
Source: Viblo Asia

- **GAN (Generative Adversarial Network):** Sinh dữ liệu mới từ nhiễu.

Source: Viblo Asia

2. **Sử dụng validation set**

- Chia training set thành training set và validation set để kiểm tra độ chính xác của model trong quá trình huấn luyện **(holdout method)**.
- Để cân bằng giữa training set và validation set (sao cho training set đủ để huấn luyện model, và validation set không quá nhỏ dẫn đến overfitting), sử dụng **K-fold Cross validation:**
    - Chia training set thành k tập con **(fold)** không có phần tử chung, có kích thước gần bằng nhau. Sau đó thực hiện **holdout method**, lấy 1 trong số k tập con làm validation **(holdout fold)**, huấn luyện model trên k-1 fold còn lại.



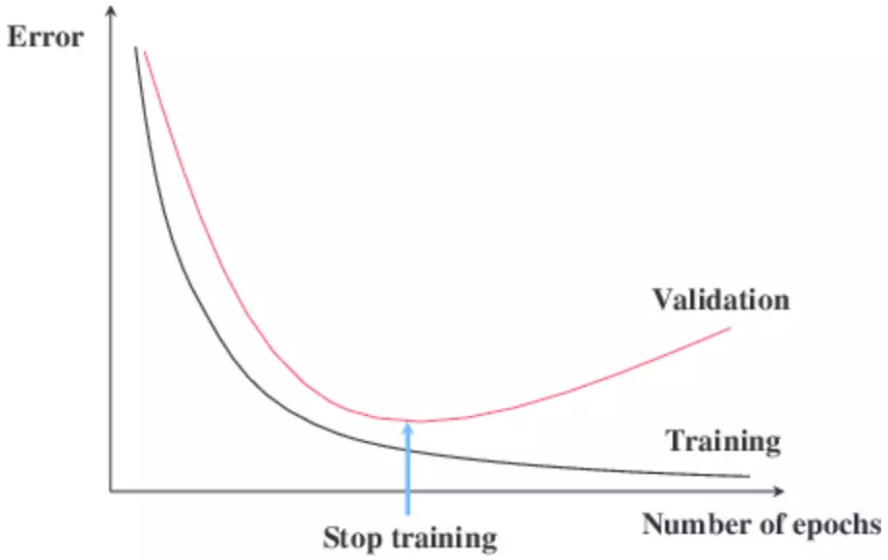
Source: Elite Data Science

=> Nhược điểm: Huấn luyện model k lần, chi phí tính toán cao.

- Khi k bằng với số phần tử trong training set ban đầu
  => **Leave one out method**

3. Regularization

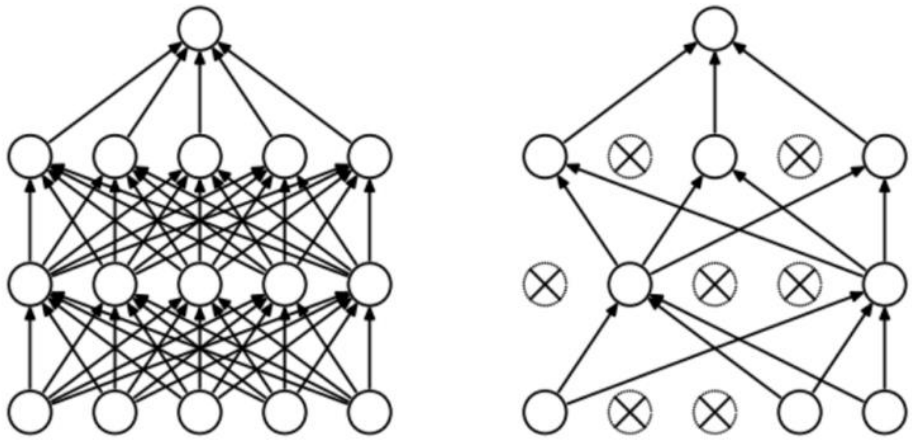
- Regularization là chuẩn hóa model trong khi vẫn giữ được tính tổng quát của nó . Các phương pháp:
  - **Early stopping:**
- Khi huấn luyện model thì không phải lúc nào (hàm mất mát) loss của training set và validation set cũng đồng thời giảm, tới một epoch nào đó thì loss của training set sẽ tiếp tục giảm nhưng loss của validation set có xu hướng tăng.
  => Dừng việc huấn luyện (vì tiếp tục huấn luyện model chỉ tốn tài nguyên).



Source: Viblo Asia

  - **Dropout:**

- Dropout với xác suất p nghĩa là trong quá trình huấn luyện model, ngẫu nhiên loại bỏ p% số lượng node trong layer nhất định, hay nói cách khác là giữ lại (1-p%) node. Mỗi layer có thể có các hệ số dropout p khác nhau.
- Ví dụ mô hình neural network 1-2-1: 1 input layer, 2 hidden layer và 1 output layer. Trên hidden layer 1, ta dùng dropout với p = 0.6 => giữ lại 2/5 node. Trên hidden layer 2, dùng dropout với p = 0.4 => giữ lại 3/5 node.



(a) Standard Neural Net (b) After applying dropout.

Source: Srivastava, Nitish, et al. ”Dropout: a simple way to prevent neural networks from overfitting”, JMLR 2014

Lưu ý:

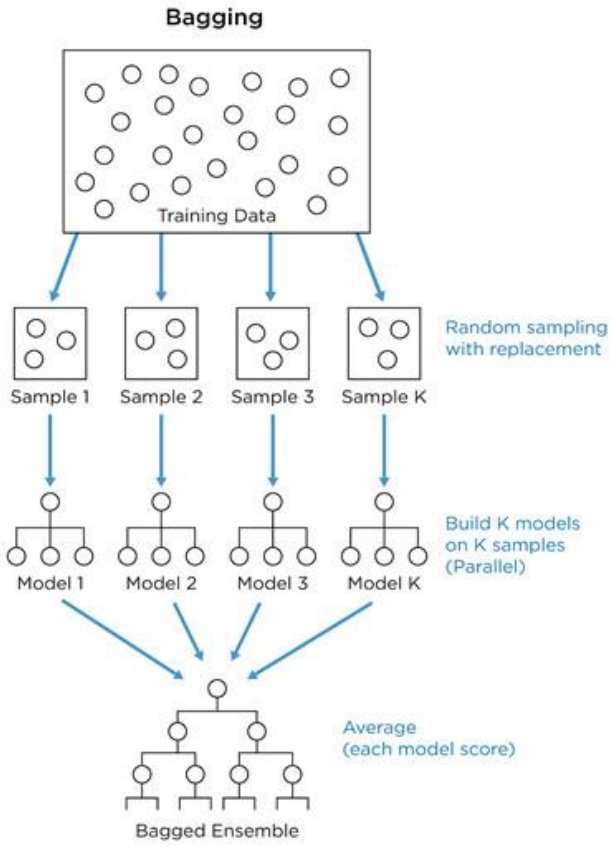
- Xác suất p nên nằm trong khoản [0.2; 0.5]; vì p quá nhỏ => không có tác dụng nhiều trong việc tránh overfitting, hoặc p quá lớn => underfitting, do các node trong layer gần như bị loại bỏ hết.
- Thường dùng dropout **trong giai đoạn huấn luyện**, đặc biệt trong Fully connected layer khi layer có quá nhiều tham số, và các node phụ thuộc mạnh mẽ vào nhau.
- Drop out có thể làm model không có tính nhất quán (consistency) do nó sẽ tắt các node một cách ngẫu nhiên dẫn đến output khác nhau (có thể tắt các node quan trọng) => Không dùng trong validation/test.

    - **Thêm “phạt” (penalty) vào hàm mất mát (loss function)**
        - **L1 Regularization - LASSO(Least Absolute Shrinkage and Selection Operator)**
        - **L2 Regularization - Ridge regression.**

4. Ensembling

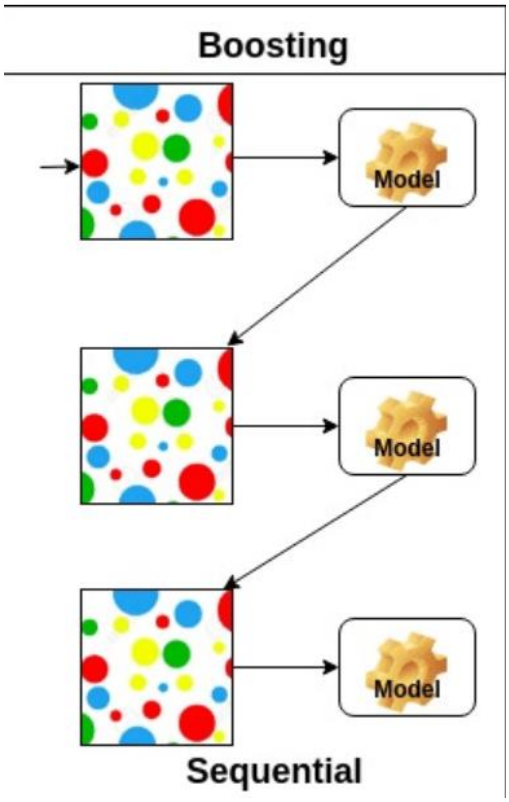
- Kết hợp các model lại với nhau để cho ra một model mạnh hơn.
- Ví dụ các model có high bias, low variance thì model tổ hợp phải có xu hướng giảm bias, ngược lại với các model có low bias, high variance.

- Các phương pháp:
    - **Bagging**: Xây dựng một lượng lớn **các model phức tạp** (thường là cùng loại) trên những subsamples khác nhau từ tập training dataset (random sample tạo 1 dataset mới). Những model này sẽ được train **độc lập và song song** với nhau nhưng đầu ra của chúng sẽ được trung bình cộng để cho ra kết quả cuối cùng.



Source: ai-blog.bappartners

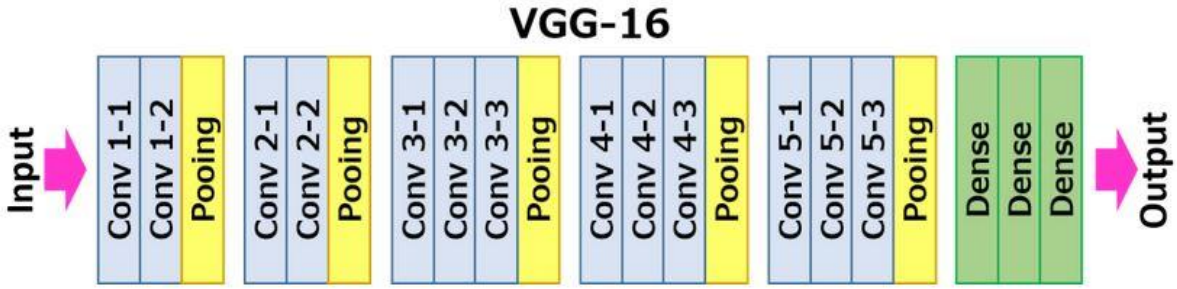
    - **Boosting**: Xây dựng một lượng lớn **các model đơn giản** (thường là cùng loại). Mỗi model sau sẽ học cách sửa những errors của model trước một cách tuần tự (dữ liệu mà model trước dự đoán sai) => Tăng (boost) độ phức tạp lên => Kết hợp tạo thành model cuối cùng mạnh nhất (vì model sau sẽ tốt hơn model trước nên tương tự kết quả sau cũng sẽ tốt hơn kết quả trước).



Source: Datacamp

Mô tả model:

Xây dựng model theo ý tưởng VGG16



Source: Machine Learning Knowledge

Model của nhóm được xây dựng dựa trên mô hình VGG16 gồm tất cả 6 khối mạng. Trong đó 5 khối đầu tiên gồm những lớp Convolution, Pooling có nhiệm vụ trích xuất đặc tính (Features extraction) của ảnh, khối cuối cùng gồm 3 lớp Fully connected có nhiệm vụ phân loại ảnh (Classification)

Mô tả chi tiết của từng lớp sẽ được thể hiện bên dưới đây.

Convention

Tất cả các lớp Convention trong model đều dùng kernel có kích thước 3x3, activate function là ReLU, và padding same để kích thước ảnh đầu ra giống với kích thước ảnh đầu vào. Tuy nhiên số lượng kernel dùng cho mỗi lớp có sự khác nhau, cụ thể như sau:

Thứ tự lớp Convolution	Số lượng kernel
1,2	64
4,5	128
7,8,9	256
11,12,13	512
15,16,17	512

Pooling

Lớp Pooling của nhóm dùng hàm max-pooling, filter có kích thước 2x2 với stride = 2. Do đó sau mỗi lần Pooling kích thước của ảnh sẽ giảm đi một nửa.

Sau khi đi qua lớp Pooling cuối cùng, dữ liệu ở dạng ma trận sẽ được duỗi thẳng (Flatten) thành kiểu vector để đưa vào các lớp Fully connected

Dense

Khối cuối cùng gồm 3 lớp Fully connected. Trong đó 2 lớp đầu tiên có độ rộng là 4096 neutron có activate function là ReLU. Tuy nhiên lớp cuối cùng chỉ có 1 neutron (vì chúng ta chỉ phân 2 loại “Chó” và “Mèo”) và activate function sử dụng là sigmoid.

Train model

Sau khi xây dựng model thành công, chúng ta phải cung cấp dữ liệu cho model để nó có thể học cách phân loại và tăng cường độ chính xác của sau mỗi lần học.

Ở bước này, chúng ta cần quan tâm đến 2 yếu tố đó là thuật toán tối ưu hóa (optimizer) và loss function của model.

Với bài toán phân loại Chó và Mèo, nhóm đã chọn

- Optimizer: **Mini-batch Gradient Descent**
- Loss function: **Binary Cross Entropy**

Ngoài ra, ta còn chọn **learning rate = 0.001** và **momentum = 0.9**

Tại sao chọn Mini-batch Gradient Descent (Mini-batch GD) ?

Thuật toán Mini-batch GD giúp giảm được độ phức tạp và thời gian tính toán do có được sự đơn giản của thuật toán Stochastic GD.

Hội tụ nhanh hơn và tránh được tình trạng cực tiểu cục bộ do việc cập nhật lại trọng số nhiều lần với các bộ dữ liệu khác nhau trong 1 epoch.

Tránh được tình trạng cực tiểu cục bộ vì có sự xáo trộn dữ liệu.

Tiết kiệm bộ nhớ bởi vì mỗi lần lặp chỉ cần tải 1 phần nhỏ của trong bộ dữ liệu.

Tại sao chọn Binary Cross Entropy?

Phân loại kết quả chỉ có 2 giá trị là 0 hoặc 1 (Mèo hoặc Chó), đây là yếu tố để nhận biết áp dụng loss function Binary Cross Entropy.

Hàm Binary Cross Entropy cho ra độ lỗi rất lớn khi có một dự đoán sai (giá trị output tiến về 0) trong khi hàm MSE cho ra độ lỗi có giá trị [0,1] cho tất cả mọi dự đoán.

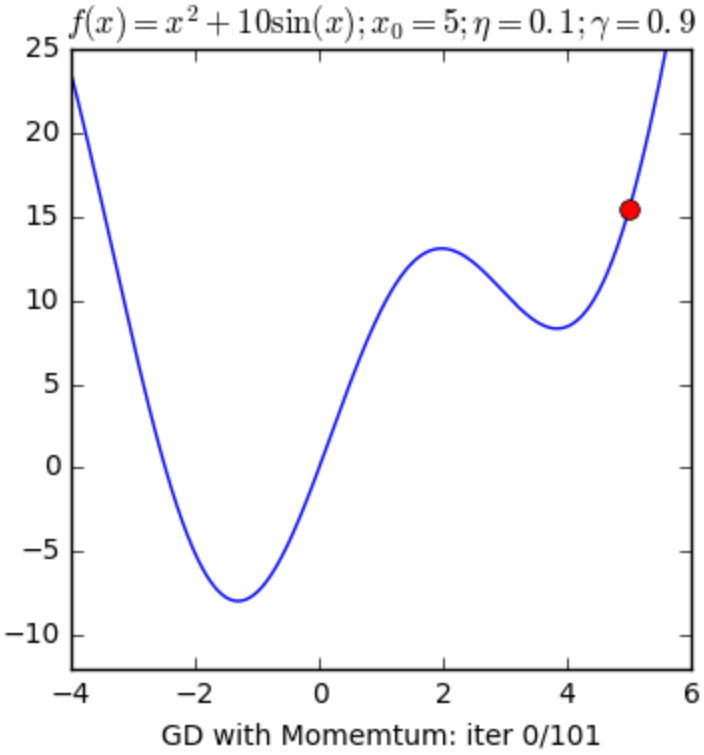
Tại sao chọn mô hình VGG16?

VGG16 là mô hình mạng Convolution được thiết kế bởi giáo sư K. Simonyan và A. Zisserman đến từ Đại học Oxford. Mô hình này đạt được độ chính xác 92.7% với bộ dữ liệu khổng lồ từ ImageNet.

Theo báo cáo "Very Deep Convolutional Networks for Large-Scale Image Recognition" của tác giả, việc sử dụng Convolution filter 3x3 làm giảm số lượng trọng số phải train đi đáng kể (giảm 81% so với filter 7x7). Ngoài ra mô hình cũng có khá nhiều lớp (21 lớp) giúp tăng độ chính xác khi dự đoán.

Tại sao phải thêm momentum?

Việc sử dụng momentum giúp ta nhảy ra khỏi local minimum khi loss function của chúng ta trở nên phức tạp.



Ưu và nhược điểm:

Ưu điểm:

- Accuracy cao, loss thấp
- Tốc độ tăng accuracy khá nhanh sau vài epochs

Nhược điểm:

- Mô hình phức tạp
- Huấn luyện model từ đầu (train from scratch) tốn nhiều thời gian.
- Weight của model khá lớn => Các vấn đề về tài nguyên (GPU, disks, ...)
- Dễ bị dự đoán nhầm mèo thành chó

Cải tiến:

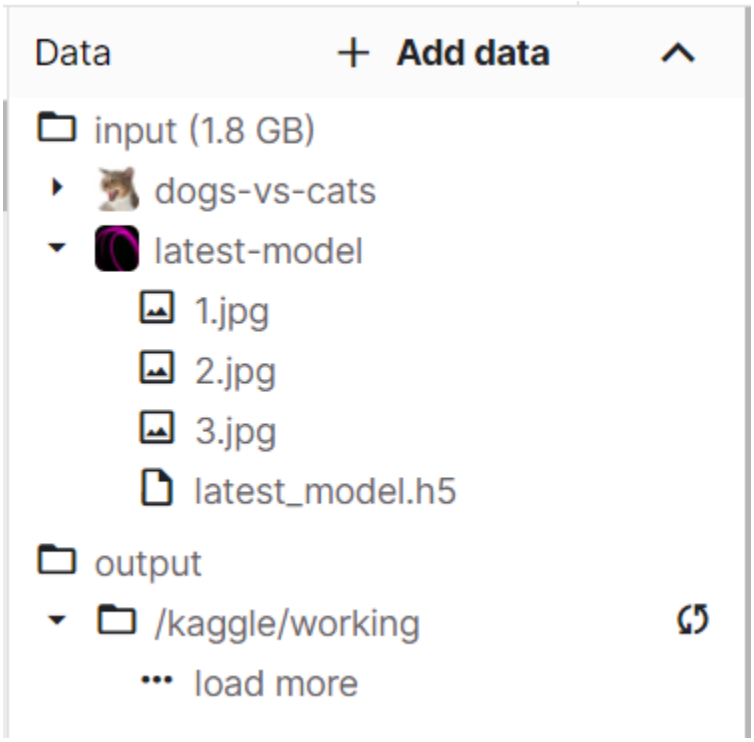
- Huấn luyện model bằng ImageDataGenerator (Tăng cường dữ liệu: phóng to, thu nhỏ ngẫu nhiên, làm méo ảnh, tăng/giảm độ sáng, …)

- Sử dụng phương pháp early stopping để có thể tiết kiệm thời gian huấn luyện mô hình.
- Sử dụng phiên bản cải tiến của VGG16 đó chính là VGG19 với số lượng các lớp trong mô hình được tăng thêm, giúp tăng độ chính xác cho mô hình.

Cách build model:

Source-code được public trên kaggle:
https://www.kaggle.com/nguynngcphnganh/image-classification-42-78-97
Đã đính kèm file input và model (latest_model.h5) với độ chính xác hơn 91% (đã train sẵn)

- **Cấu trúc Input:** Dữ liệu data gồm 25000 ảnh trên kaggle từ cuộc thi dogs-vs-cats classification trên kaggle đã được import sẵn.



- Chạy **cell đầu tiên** để extract files từ input và sử dụng nếu chưa thấy folder <train> và <test1> ở trong /output/kaggle/working:

Unzip file dữ liệu từ kaggle gồm 25000 ảnh chó mèo bằng cách tham gia cuộc thi dogs-vs-cats

```
import os
base_dir = "../input/dogs-vs-cats/"

train_link = os.path.join(base_dir, "train.zip")
test_link = os.path.join(base_dir, "test1.zip")

# ../input/dogs-vs-cats-redux-kernels-edition/test.zip
import zipfile
with zipfile.ZipFile(train_link,"r") as z:
 z.extractall()
with zipfile.ZipFile(test_link,"r") as z:
 z.extractall()
```

- Lần lượt chạy các bước sau để chuẩn bị dataset và model (có chú thích trong code)

- Bước 1: Import thư viện
- Bước 2: Xác định các thuộc tính hình ảnh - Kích thước, chiều
- Bước 3: Chuẩn bị bộ dữ liệu
- Bước 4: Định nghĩa callbacks và learning rate
- Bước 5: Quản lý dữ liệu - chia tập validation và train
- Bước 6: Generate tập dữ liệu và điều chỉnh

- Sau đó, ta có 2 lựa chọn: **Build một model hoàn toàn mới** hoặc **Load model nhóm đã build và train sẵn.** Với lựa chọn tạo model mới:

- Bước 7: Build model khi CHƯA CÓ MODEL
- Bước 8: Compile và đánh giá
- Bước 9: Fit model (training)

- Còn với lựa chọn load model đã có sẵn:

- Bước 10: Load lại model
- Có thể chạy lại Bước 9 nếu muốn train thêm, hoặc đơn giản chỉ để xem độ chính xác của model hiện tại (Cancel run sau khi đã đọc được accuracy)

- Sau khi đã có model, ta sẽ bắt đầu dự đoán kết quả (ảnh là chó hay mèo):

- Bước 11: Chuẩn bị dữ liệu thử nghiệm
- Bước 12: Dự đoán kết quả trên 1 số ảnh. Có thể thay đổi điều kiện dừng để giới hạn số ảnh in ra:

```
if(cat_counter==10): break
```

Kết quả dự đoán: 0 - mèo, 1 - chó.

- Hỗ trợ thêm 2 hàm:

- Bước 13: Test 1 ảnh truyền vào, thay đổi dòng cuối (chứa đường dẫn chứa tấm ảnh muốn test). Với điều kiện tấm ảnh đã nằm trong input data của project

```
predict_("../input/testcase/con_meo_cua_truc.jpg")
```

- Bước 14: Đánh giá tỉ lệ chính xác - sau khi train N epochs: Vẽ biểu đồ thể hiện sự biến thiên của accuracy và loss sau từng vòng train. Hàm chỉ sử dụng cho model ĐÃ chạy bước 9 ít nhất 1 lần (đã train) chứ không sử dụng được cho model load thẳng vào và chưa train.

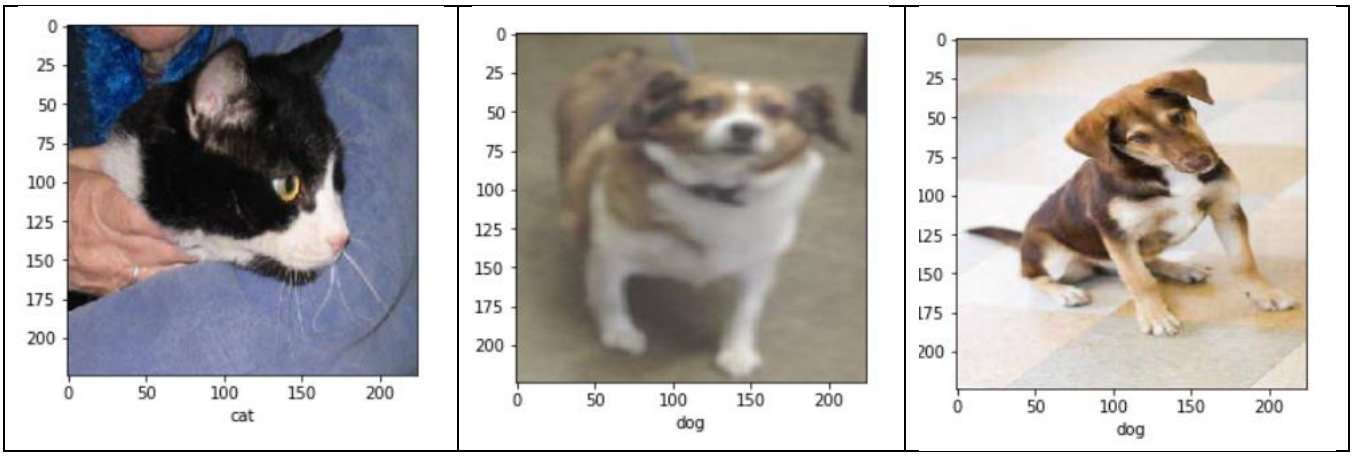
Kết quả:

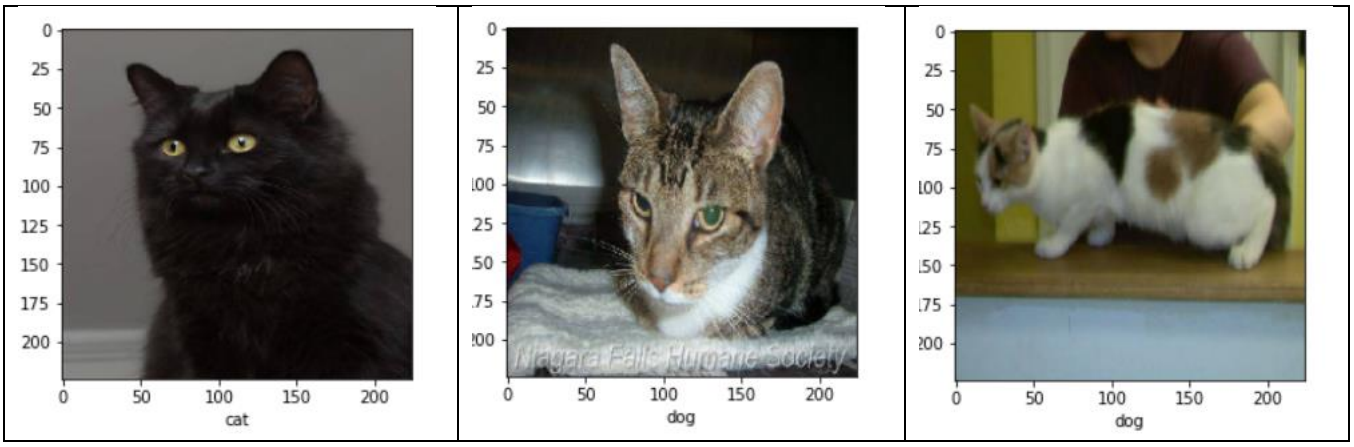
- Sau khi load model đã train ở Bước 10, chạy thử Bước 9, train model lại hai lần để kiểm tra loss, accuracy, val_loss, val_accuracy của model.

```
#Bước 9: Fit model (training) - CHẠY KHI LOAD MODEL để kiểm tra accuracy hiện tại rồi tắt đi
history = model.fit_generator(generator=train_generator,
                              steps_per_epoch=len(train_generator),
                              validation_data=validation_generator,
                              validation_steps=len(validation_generator),
                              epochs=1,
                              verbose=1,
                              callbacks=callbacks)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
586/586 [==============================] - 320s 528ms/step - loss: 0.1516 - accuracy: 0.9371 - val_loss: 0.1615 - val_accuracy: 0.9320
```

- Chạy model trên tập test, in ra một số ảnh ngẫu nhiên sau khi được model gán nhãn ở Bước 12:

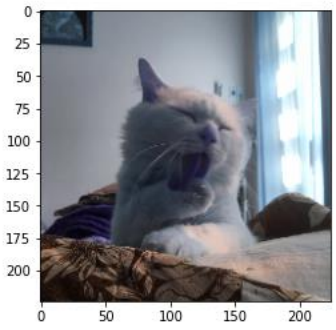




- Kiểm tra model trên 1 bức ảnh bất kì

```
#Bước 13: Test 1 ảnh truyền vào - CHẠY NẾU MUỐN TEST 1 ẢNH = TAY
def predict_(link):
    image = cv2.imread(link)
    image = cv2.resize(image, (224,224))
    preds = model.predict(np.expand_dims(image, axis=0))[0]
    plt.imshow(image)
    plt.show()
    if preds==0:
            print("Predicted Label:Cat")
    else:
            print("Predicted Label: Dog")


predict_("../input/latest-model/2.jpg")
```



```
Predicted Label:Cat
```

REFERENCES:

1. Các khái niệm
2. Bias vs Variance Tradeoff (elitedatascience.com)
3. Các phương pháp phòng tránh Overfitting:
    a. elitedatascience.com
    b. CFI
    c. viblo.asia
    d. machinelearningcoban.com
    e. Dropout (nttuan8)
        i. Tại sao không dùng drop out trong validation và test
    f. Ensembling
        i. viblo.asia
        ii. AI blog.bappartners
4. Xây dựng model
    a. Cấu hình Keras VGG16
    b. VGG16 (2)
    c. Xây dựng input
5. Mô tả model
    a. Optimizer (viblo.asia)
    b. Mini-batch Gradient Descent
        i. machinelearningmastery.com
        ii. machinelearningcoban.com
        iii. ruder.io
    c. Binary cross entropy (Youtube)