Học sâu là một vấn đề con của học máy, no học các biểu diễn học tập của dữ liệu. Đặc biệt hiệu quả trong việc học các mẫu. Các thuật toán học sâu cố gắng học (nhiều cấp độ) biểu diễn bằng cách sử dụng hệ thống phân cấp gồm nhiều lớp/tầng.

Refresh

1. Multilayer Perceptron

1.1. Import thư viện

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from importlib import reload
import seaborn as sns
import random
random.seed(10)
# train test split
from sklearn.model_selection import train_test_split

# scaler
from sklearn.preprocessing import MinMaxScaler
```

```
# machine learning
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# deep learning
from keras import models
from keras import layers
from keras.utils import to_categorical

# metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix
```

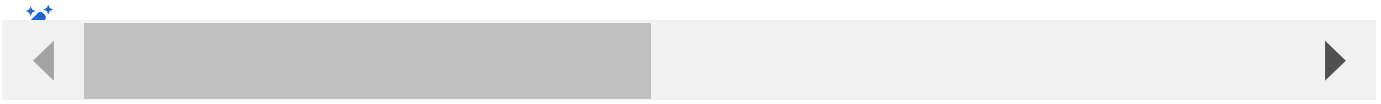
1.2. Load dữ liệu

```
path = '/content/drive/MyDrive/Colab Notebooks/VT-DAC/data/'
fruit_train = pd.read_csv(path+'train.csv')
fruit_test = pd.read_csv(path+'test.csv')
```

```
fruit_train.head()
```

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA	EXTENT	ASPECT_RATI
0	298963	2119.7129	722.8275	544.2319	0.6581	616.9696	0.9424	317248	0.6671	1.328
1	159802	1530.7870	564.7152	362.0923	0.7674	451.0723	0.9920	161095	0.6801	1.559
2	419881	2440.1890	843.5652	645.4122	0.6439	731.1697	0.9893	424423	0.7533	1.307
3	376389	2409.2590	941.2642	509.9813	0.8405	692.2668	0.9852	382055	0.7680	1.845
4	451414	2457.2080	889.3253	652.4316	0.6796	758.1281	0.9970	452755	0.7877	1.363

5 rows × 35 columns



```
X_fruit_train_raw = fruit_train.drop('Class',axis=1)
y_fruit_train = fruit_train['Class']

X_fruit_test_raw = fruit_test.drop('Class',axis=1)
y_fruit_test = fruit_test['Class']
```

```
print('training shape', X_fruit_train_raw.shape)
print('testing shape', X_fruit_test_raw.shape)
```

```
training shape (718, 34)
testing shape (180, 34)
```

1.3. Hàm đánh giá kết quả

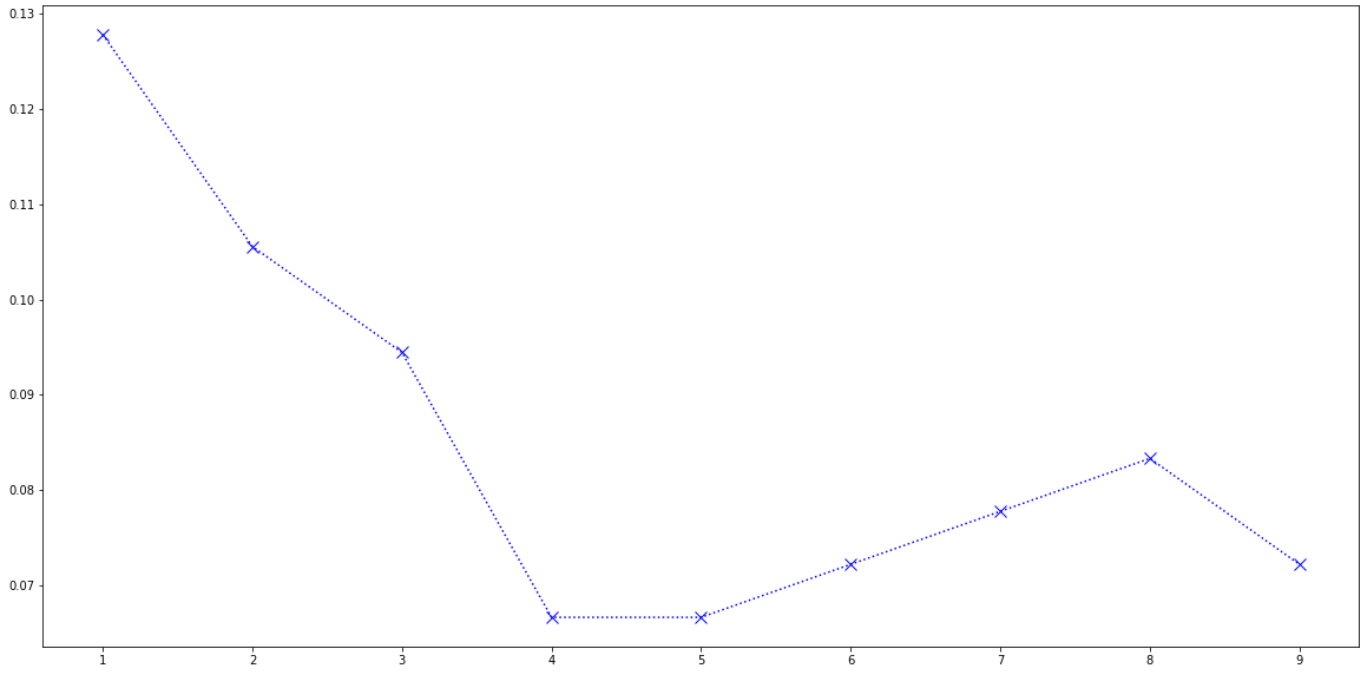
```
# Tính các độ đo đánh giá mô hình
def evaluate(y_test, y_pred):
```

```
    acc = accuracy_score(y_test, y_pred)
    print("ACC: ", acc)
    print(classification_report(y_test,y_pred))


# Trực quan hoá một số thông tin
def visualize_result(y_test, y_pred):
    sns.heatmap(confusion_matrix(y_test,y_pred),cmap='Blues',annot=True,linewidths=2,linecolor='white')
```

1.4. Phân lớp thông thường với Random forest classifier

Trong phần đầu tiên này, chúng ta sẽ quay lại vấn đề phân loại trái cây bằng cách sử dụng các tính năng được cung cấp trong tập dữ liệu hướng dẫn lần trước.

```
err = []

for i in range(1,10):
    rfe = RandomForestClassifier(n_estimators=i*10)
    rfe.fit(X_fruit_train_raw,y_fruit_train)
    errpred = rfe.predict(X_fruit_test_raw)
    err.append(np.mean(errpred != y_fruit_test))

plt.figure(figsize=(20,10))
plt.plot(range(1,10),err,color='blue',linestyle='dotted',marker='x',markerfacecolor='red',markersize=10)
plt.title = 'Number of estimators VS Error Rates'
plt.xlabel = 'Estimators'
plt.ylabel= 'Error Rate'
plt.show()
```

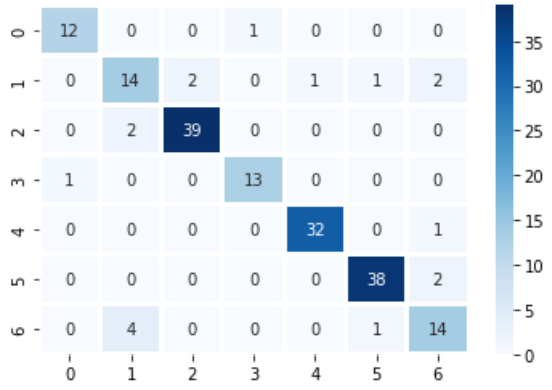


```
rfc = RandomForestClassifier(n_estimators=50,max_features='auto',max_depth=8)
rfc.fit(X_fruit_train_raw,y_fruit_train)
```

```
RandomForestClassifier(max_depth=8, n_estimators=50)
```

```
y_rfc_pred = rfc.predict(X_fruit_test_raw)
evaluate(y_fruit_test, y_rfc_pred)
visualize_result(y_fruit_test, y_rfc_pred)
```

```
ACC:  0.9
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        13
           1       0.70      0.70      0.70        20
           2       0.95      0.95      0.95        41
           3       0.93      0.93      0.93        14
           4       0.97      0.97      0.97        33
           5       0.95      0.95      0.95        40
           6       0.74      0.74      0.74        19

    accuracy                           0.90       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.90      0.90      0.90       180
```



1.5. Neural Networks

Tiếp theo, chúng ta Multilayer perceptron trên cùng một tập dữ liệu để xem nó hoạt động tốt như thế nào.

Đầu tiên, chúng ta định nghĩa kiến trúc của mô hình, chỉ chứa 1 lớp đầu vào, 1 lớp ẩn và 1 lớp đầu ra.

```
# Scaling the data using standard scaler
scaler = MinMaxScaler()
scaler.fit(X_fruit_train_raw)
X_fruit_train = scaler.transform(X_fruit_train_raw)
X_fruit_test = scaler.transform(X_fruit_test_raw)

print('training shape', X_fruit_train.shape)
print('testing shape', X_fruit_train.shape)
```

```
training shape (718, 34)
testing shape (718, 34)
```

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(34,)))
network.add(layers.Dense(7, activation='softmax'))
```

Ở đây, mạng nơ ron của chúng ta bao gồm một chuỗi gồm hai lớp fully-connected, là các lớp thần kinh được kết nối dày đặc. Lớp thứ hai (cuối cùng) là lớp "softmax" 7 chiều, có nghĩa là nó sẽ trả về một mảng gồm 7 điểm xác suất (tổng bằng 1). Mỗi điểm sẽ là xác suất mà ví dụ hiện tại thuộc về một trong 7 loại trái cây.

Để mạng nơ ron sẵn sàng huấn luyện chúng ta cần định nghĩa các thành phần sau đây:

- Hàm mất mát: là cách đo lường mức độ hiệu quả của mô hình trên dữ liệu huấn luyện, và làm thế nào để mạng tối ưuưu đúng hướng.
- Tối ưu hóa: đây là cơ chế mà qua đó mô hình học sâu sẽ tự cập nhật tham số dựa trên dữ liệu mà mô hình học sâu nhìn thấy thông qua hàm mất mát.
- Hàm đánh giá để theo dõi trong quá trình huấn luyện và thử nghiệm. Ở đây chúng ta sẽ sử dụng độ chính xác.

```
network.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
network.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               17920

 dense_1 (Dense)             (None, 7)                 3591

=================================================================
Total params: 21,511
Trainable params: 21,511
Non-trainable params: 0
_________________________________________________________________
```

Chúng ta cần mã hóa nhãn về dạng one-hot.

```
train_labels = to_categorical(y_fruit_train)
```

Bây giờ chúng ta đã sẵn sàng để huấn luyện, điều này trong Keras được thực hiện thông qua một cuộc gọi đến phương thức phù hợp của mạng: chúng ta "fit" mô hình với dữ liệu đào tạo của nó.

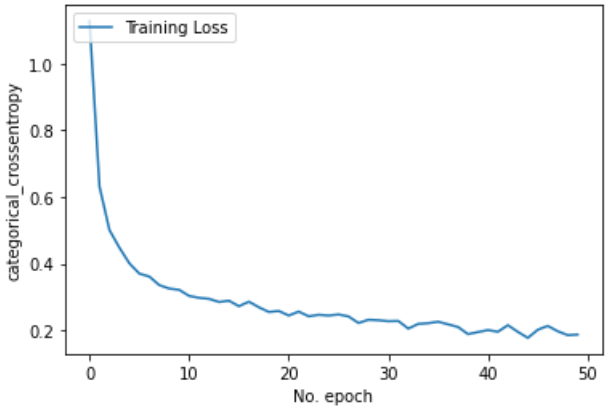
```
history = network.fit(X_fruit_train, train_labels, epochs=50, batch_size=2)
```

```
Epoch 22/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2563 - accuracy: 0.8969
Epoch 23/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2421 - accuracy: 0.8997
Epoch 24/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2467 - accuracy: 0.9039
Epoch 25/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2445 - accuracy: 0.9081
Epoch 26/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2477 - accuracy: 0.8844
Epoch 27/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2418 - accuracy: 0.9025
Epoch 28/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2220 - accuracy: 0.9136
Epoch 29/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2317 - accuracy: 0.9109
Epoch 30/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2303 - accuracy: 0.9067
Epoch 31/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2275 - accuracy: 0.9164
Epoch 32/50
```

```
359/359 [==============================] - 1s 2ms/step - loss: 0.2282 - accuracy: 0.9109
Epoch 33/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2052 - accuracy: 0.9136
Epoch 34/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2194 - accuracy: 0.9150
Epoch 35/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2215 - accuracy: 0.9081
Epoch 36/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2258 - accuracy: 0.9025
Epoch 37/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2181 - accuracy: 0.9164
Epoch 38/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2100 - accuracy: 0.9136
Epoch 39/50
359/359 [==============================] - 1s 3ms/step - loss: 0.1887 - accuracy: 0.9220
Epoch 40/50
359/359 [==============================] - 1s 3ms/step - loss: 0.1950 - accuracy: 0.9192
Epoch 41/50
359/359 [==============================] - 1s 3ms/step - loss: 0.2008 - accuracy: 0.9136
Epoch 42/50
359/359 [==============================] - 1s 2ms/step - loss: 0.1958 - accuracy: 0.9192
Epoch 43/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2155 - accuracy: 0.9136
Epoch 44/50
359/359 [==============================] - 1s 2ms/step - loss: 0.1955 - accuracy: 0.9192
Epoch 45/50
359/359 [==============================] - 1s 2ms/step - loss: 0.1773 - accuracy: 0.9331
Epoch 46/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2017 - accuracy: 0.9150
Epoch 47/50
359/359 [==============================] - 1s 2ms/step - loss: 0.2134 - accuracy: 0.9067
Epoch 48/50
359/359 [==============================] - 1s 2ms/step - loss: 0.1973 - accuracy: 0.9234
Epoch 49/50
359/359 [==============================] - 1s 2ms/step - loss: 0.1857 - accuracy: 0.9150
Epoch 50/50
359/359 [==============================] - 1s 2ms/step - loss: 0.1873 - accuracy: 0.9276
```

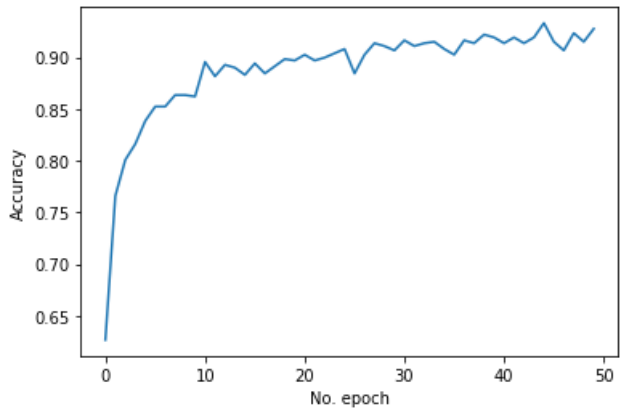
Visualize the training loss

```
plt=reload(plt)
plt.plot(history.history["loss"], label="Training Loss")
plt.ylabel("categorical_crossentropy")
plt.xlabel("No. epoch")
plt.legend(loc="upper left")
plt.show()
```



Visualize the training accuracy

```
plt=reload(plt)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('No. epoch')
plt.show()
```

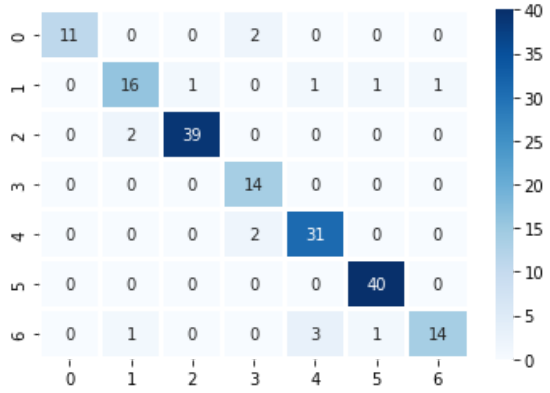



```
nw_pred = network.predict(X_fruit_test)
nw_pred=np.argmax(nw_pred,axis=1)
```

```
6/6 [==============================] - 0s 2ms/step
```

```
evaluate(y_fruit_test, nw_pred)
visualize_result(y_fruit_test, nw_pred)
```

```
ACC:  0.9166666666666666
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        13
           1       0.84      0.80      0.82        20
           2       0.97      0.95      0.96        41
           3       0.78      1.00      0.88        14
           4       0.89      0.94      0.91        33
           5       0.95      1.00      0.98        40
           6       0.93      0.74      0.82        19

    accuracy                           0.92       180
   macro avg       0.91      0.90      0.90       180
weighted avg       0.92      0.92      0.92       180
```



1.6. Huấn luyện với tập dữ liệu lớn hơn

Vấn đề tiếp theo mà chúng ta giải quyết ở đây là phân loại các hình ảnh thang độ xám của các chữ số viết tay (28 pixel x 28 pixel), thành 10 danh mục của chúng (0 đến 9). Tập dữ liệu mà chúng ta sẽ sử dụng là tập dữ liệu MNIST, một tập dữ liệu cổ điển trong cộng đồng máy học, đã tồn tại gần như cùng với lĩnh vực này và đã được nghiên cứu rất chuyên sâu. Đó là một bộ gồm 60.000 hình ảnh đào tạo, cộng với 10.000 hình ảnh thử nghiệm, do Viện Tiêu chuẩn và Công nghệ Quốc gia (NISTT) xây dựng vào những năm 1980.

Bộ dữ liệu MNIST được tải sẵn trong Keras, dưới dạng một bộ gồm bốn mảng Numpy:

```
from keras.datasets import mnist
```

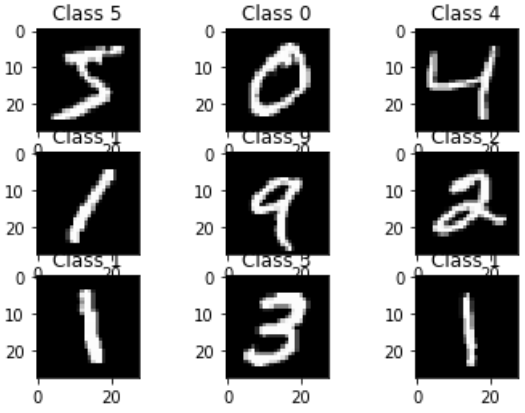
Trước khi huấn luyện, chúng ta sẽ xử lý trước dữ liệu bằng cách định hình lại dữ liệu thành hình dạng mà mạng mong đợi và chia tỷ lệ dữ liệu sao cho tất cả các giá trị đều nằm trong khoảng [0, 1]. Ví dụ, trước đây, hình ảnh đào tạo được lưu trữ trong một mảng có hình dạng (60000, 28, 28) thuộc loại uint8 với các giá trị trong khoảng [0, 255]. Chúng ta chuyển đổi nó thành một mảng float32 có hình dạng (60000, 28 * 28) với các giá trị từ 0 đến 1.

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print(train_images.shape)
print(test_images.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
(60000, 28, 28)
(10000, 28, 28)
```

Visualize MNIST dataset

```
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(train_images[i], cmap='gray', interpolation='none')
    plt.title("Class {}".format(train_labels[i]))
```




```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

K-Nearest Neighbors

```
knn = KNeighborsClassifier()
```

```
knn.fit(train_images[:1000], train_labels[:1000])
print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(train_images[:500], train_labels[:500])))
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(test_images[:100], test_labels[:100])))
```

```
Accuracy of K-NN classifier on training set: 0.91
Accuracy of K-NN classifier on test set: 0.84
```

```
knn_pred = knn.predict(test_images)
evaluate(test_labels, knn_pred)
```

```
ACC:  0.8582
              precision    recall  f1-score   support

           0       0.90      0.97      0.93       980
           1       0.80      0.99      0.89      1135
           2       0.93      0.79      0.85      1032
           3       0.88      0.85      0.87      1010
           4       0.85      0.79      0.82       982
           5       0.84      0.79      0.82       892
           6       0.91      0.92      0.91       958
           7       0.86      0.88      0.87      1028
           8       0.93      0.71      0.81       974
           9       0.75      0.86      0.80      1009

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

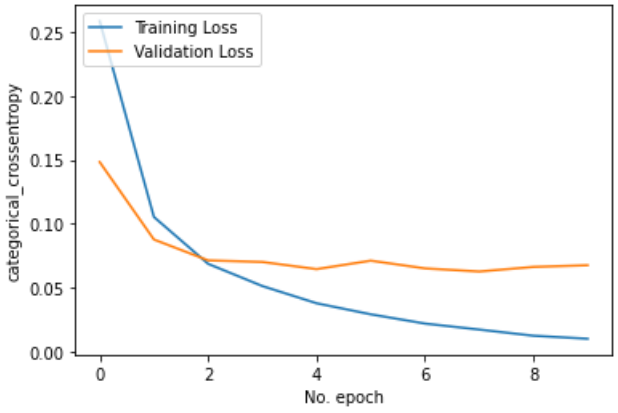
Neural Networks

```
y_train = to_categorical(train_labels)
y_test = train_labels = to_categorical(test_labels)
```

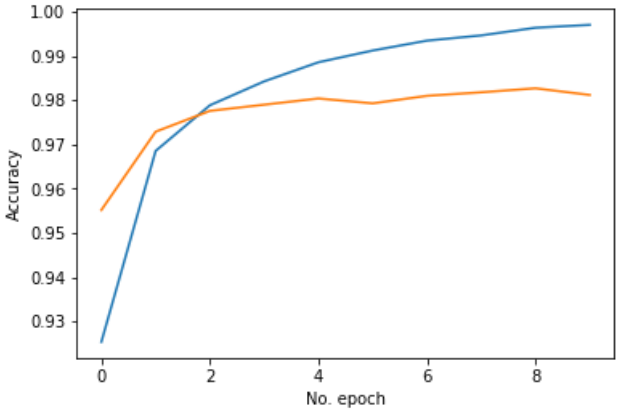
```
network2 = models.Sequential()
network2.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network2.add(layers.Dense(10, activation='softmax'))

network2.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

history = network2.fit(train_images, y_train, epochs=10, batch_size=128, validation_data=(test_images, y_test))
```

```
Epoch 1/10
469/469 [==============================] - 2s 4ms/step - loss: 0.2584 - accuracy: 0.9254 - val_loss: 0.1483 - v
Epoch 2/10
469/469 [==============================] - 2s 4ms/step - loss: 0.1054 - accuracy: 0.9686 - val_loss: 0.0878 - v
Epoch 3/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0688 - accuracy: 0.9789 - val_loss: 0.0715 - v
Epoch 4/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0514 - accuracy: 0.9843 - val_loss: 0.0703 - v
Epoch 5/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0380 - accuracy: 0.9886 - val_loss: 0.0647 - v
Epoch 6/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0294 - accuracy: 0.9912 - val_loss: 0.0713 - v
Epoch 7/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0221 - accuracy: 0.9935 - val_loss: 0.0652 - v
Epoch 8/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0174 - accuracy: 0.9947 - val_loss: 0.0628 - v
Epoch 9/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0126 - accuracy: 0.9964 - val_loss: 0.0664 - v
Epoch 10/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0103 - accuracy: 0.9970 - val_loss: 0.0677 - v
```

```
plt=reload(plt)
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.ylabel("categorical_crossentropy")
plt.xlabel("No. epoch")
plt.legend(loc="upper left")
plt.show()
```




```
plt=reload(plt)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('No. epoch')
plt.show()
```




```
img_pred = network2.predict(test_images)
img_pred=np.argmax(img_pred,axis=1)
```

```
313/313 [==============================] - 1s 2ms/step
```

```
evaluate(test_labels, img_pred)
```

```
ACC:  0.9812
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
```

```
              4       0.98      0.98      0.98       982
              5       0.97      0.98      0.98       892
              6       0.99      0.98      0.98       958
              7       0.98      0.98      0.98      1028
              8       0.97      0.98      0.98       974
              9       0.98      0.97      0.97      1009

       accuracy                           0.98     10000
      macro avg       0.98      0.98      0.98     10000
   weighted avg       0.98      0.98      0.98     10000
```

2. Phân loại ảnh với CNN

Phân loại hình ảnh của hoa.

2.1. Import thư viện

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
import pathlib
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

2.2. Down load tập dữ liệu

Bộ dữ liệu bao gồm gần 3700 ảnh và có cấu trúc như sau:

```
flower_photo/
  daisy/
  dandelion/
  roses/
  sunflowers/
  tulips/
```

```
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)
```

Sau khi tải xuống, chúng ta có một bản sao của tập dữ liệu. Có tổng số 3.670 hình ảnh:

```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```
3670
```

Đây là hoa hồng

```
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[1]))
```

```
PIL.Image.open(str(roses[5]))
```

Đây là hoa tulips:

```
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[1]))
```

```

```

```
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[5]))
```

2.3. Tiền xử lý dữ liệu với keras

Tạo tập dữ liệu

```
batch_size = 4
img_height = 180
img_width = 180
```

```
train_ds = keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)
```

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

Trực quan hóa dữ liệu

```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(4):
```

```
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


```
for image_batch, labels_batch in train_ds:
    print(image_batch.shape)
    print(labels_batch.shape)
    break
```

```
(4, 180, 180, 3)
(4,)
```

image_batch là một tensor có hình dạng (32, 180, 180, 3). Đây là lô gồm 32 hình ảnh có hình dạng 180x180x3 (kích thước cuối cùng đề cập đến các kênh màu RGB). label_batch là một tensor của hình (32,), đây là các nhãn tương ứng với 32 hình ảnh.

Bạn có thể gọi .numpy() trên các tensor image_batch và label_batch để chuyển đổi chúng thành numpy.ndarray.

Cấu hình tập dữ liệu

Hãy đảm bảo sử dụng tính năng tìm nạp trước vào bộ đệm để có thể tạo dữ liệu từ đĩa mà không bị chặn I/O. Đây là hai phương pháp quan trọng bạn nên sử dụng khi tải dữ liệu.

Dataset.cache() giữ hình ảnh trong bộ nhớ sau khi chúng được tải ra khỏi đĩa trong epoch đầu tiên. Điều này sẽ đảm bảo tập dữ liệu không trở thành nút thắt cổ chai trong khi huấn luyện mô hình của bạn. Nếu tập dữ liệu của bạn quá lớn để vừa với bộ nhớ, bạn cũng có thể sử dụng phương pháp này để tạo bộ nhớ đệm trên đĩa có hiệu suất cao.

Dataset.prefetch() chồng chéo quá trình tiền xử lý dữ liệu và thực thi mô hình trong khi đào tạo.

Độc giả quan tâm có thể tìm hiểu thêm về cả hai phương pháp, cũng như cách lưu trữ dữ liệu vào đĩa trong hướng dẫn hiệu suất dữ liệu.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Chuẩn hóa và biểu diễn dữ liệu

Các giá trị RGB nằm trong phạm vi [0, 255]. Điều này không lý tưởng cho mạng học sâu; nói chung, nên tìm cách làm cho giá trị đầu vào của mình nhỏ. Tại đây, sẽ chuẩn hóa các giá trị nằm trong phạm vi [0, 1] bằng cách sử dụng lớp Thay đổi tỷ lệ.

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

Lưu ý: Các lớp và tiện ích Tiền xử lý máy ảnh được giới thiệu trong phần này hiện đang thử nghiệm và có thể thay đổi.

Có hai cách để sử dụng lớp này. Bạn có thể áp dụng nó vào tập dữ liệu bằng cách gọi bản đồ:

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

Hoặc, bạn có thể bao gồm lớp bên trong định nghĩa mô hình của mình, điều này có thể đơn giản hóa việc triển khai.

Lưu ý: trước đây bạn đã thay đổi kích thước hình ảnh bằng cách sử dụng đối số image_size của image_dataset_from_directory. Nếu muốn bao gồm logic thay đổi kích thước trong mô hình của mình, có thể sử dụng Resizing layer.link text

2.4. Mô hình

Định nghĩa mô hình

Mô hình bao gồm ba tầng CNN với maxpool trong mỗi lớp. Có tầng FC 128 đơn vị ở trên và được kích hoạt bằng relu. Mô hình này chưa được điều chỉnh để có độ chính xác cao, mục tiêu của hướng dẫn này là đưa ra một cách tiếp cận baseline.

```
num_classes = 5

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

```
model.summary()
```

```
Model: "sequential_2"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten (Flatten)           (None, 30976)             0

 dense_4 (Dense)             (None, 128)               3965056

 dense_5 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

Compile mô hình

Trình tối ưu hóa Adam và hàm mất mát SparseCategoricalCrossentropy. Để xem độ chính xác của quá trình đào tạo và xác thực cho từng giai đoạn đào tạotạo.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Huấn luyện mô hình

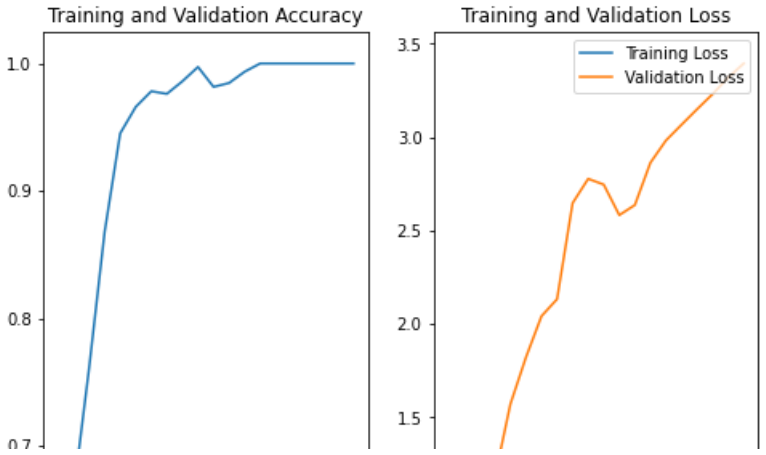
```
epochs=20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 1/20
734/734 [==============================] - 7s 7ms/step - loss: 1.2236 - accuracy: 0.5027 - val_loss: 1.0206 - v
Epoch 2/20
734/734 [==============================] - 4s 6ms/step - loss: 0.8857 - accuracy: 0.6628 - val_loss: 0.8875 - v
Epoch 3/20
734/734 [==============================] - 4s 6ms/step - loss: 0.6265 - accuracy: 0.7609 - val_loss: 1.0609 - v
Epoch 4/20
734/734 [==============================] - 4s 6ms/step - loss: 0.3533 - accuracy: 0.8678 - val_loss: 1.1779 - v
Epoch 5/20
734/734 [==============================] - 4s 6ms/step - loss: 0.1775 - accuracy: 0.9452 - val_loss: 1.5711 - v
Epoch 6/20
734/734 [==============================] - 4s 6ms/step - loss: 0.1119 - accuracy: 0.9659 - val_loss: 1.8215 - v
Epoch 7/20
734/734 [==============================] - 4s 6ms/step - loss: 0.0710 - accuracy: 0.9782 - val_loss: 2.0422 - v
Epoch 8/20
```

```
734/734 [==============================] - 4s 6ms/step - loss: 0.0825 - accuracy: 0.9762 - val_loss: 2.1332 - v
Epoch 9/20
734/734 [==============================] - 4s 6ms/step - loss: 0.0499 - accuracy: 0.9860 - val_loss: 2.6476 - v
Epoch 10/20
734/734 [==============================] - 4s 6ms/step - loss: 0.0110 - accuracy: 0.9973 - val_loss: 2.7776 - v
Epoch 11/20
734/734 [==============================] - 4s 6ms/step - loss: 0.0572 - accuracy: 0.9816 - val_loss: 2.7467 - v
Epoch 12/20
734/734 [==============================] - 5s 6ms/step - loss: 0.0587 - accuracy: 0.9847 - val_loss: 2.5831 - v
Epoch 13/20
734/734 [==============================] - 4s 6ms/step - loss: 0.0269 - accuracy: 0.9935 - val_loss: 2.6371 - v
Epoch 14/20
734/734 [==============================] - 4s 6ms/step - loss: 9.5764e-04 - accuracy: 1.0000 - val_loss: 2.8637
Epoch 15/20
734/734 [==============================] - 4s 6ms/step - loss: 1.4297e-04 - accuracy: 1.0000 - val_loss: 2.9830
Epoch 16/20
734/734 [==============================] - 4s 6ms/step - loss: 7.3277e-05 - accuracy: 1.0000 - val_loss: 3.0673
Epoch 17/20
734/734 [==============================] - 4s 6ms/step - loss: 4.6970e-05 - accuracy: 1.0000 - val_loss: 3.1503
Epoch 18/20
734/734 [==============================] - 4s 6ms/step - loss: 3.1174e-05 - accuracy: 1.0000 - val_loss: 3.2314
Epoch 19/20
734/734 [==============================] - 4s 6ms/step - loss: 2.0361e-05 - accuracy: 1.0000 - val_loss: 3.3199
Epoch 20/20
734/734 [==============================] - 4s 6ms/step - loss: 1.3692e-05 - accuracy: 1.0000 - val_loss: 3.3953
```

2.5. Đánh giá mô hình

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Như bạn có thể thấy từ các biểu đồ, độ chính xác training và độ chính xác validation bị sai lệch lớn và mô hình chỉ đạt được độ chính xác khoảng 60% trên tập hợp xác thực.

Hãy xem điều gì đã xảy ra và cố gắng tăng hiệu suất tổng thể của mô hình.

Overfitting

Trong các biểu đồ ở trên, độ chính xác training tăng tuyến tính theo thời gian, trong khi độ chính xác validation giảm khoảng 60% trong quá trình đào tạo. Ngoài ra, sự khác biệt về độ chính xác là điều đáng chú ý—một dấu hiệu của việc overfitting.

Khi tập dữ liệu huấn luyện nhỏ, mô hình đôi khi học từ nhiễu hoặc các chi tiết không mong muốn, nó tác động tiêu cực đến hiệu suất của mô hình trên các ví dụ mới. Hiện tượng này được gọi là overfitting. Điều đó có nghĩa là mô hình sẽ gặp khó khăn khi khái quát hóa trên một tập dữ liệu mới.

Có nhiều cách để giải quyết vấn đề này. VD như thêm Dropout vào mô hình.

2.6. Cải tiến mô hình

Data augmentation

Một trong những phương pháp khác để khắc phụ việc tập dữ liệu huấn luyện nhỏ ở trong xử lý ảnh là việc tăng cường dữ liệu bằng việc xử dụng các phép biến đổi ngẫu nhiên hình ảnh. Điều này giúp mô hình nhìn nhận sự việc từ nhiều khía cạnh hơn và có tính khái quát tốt hơn.

Có thể triển khai việc tăng cường dữ liệu bằng thư viện keras. Chúng được đưa và bên trong mô hình như các tầng khác và chạy trên GPU.

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(img_height, img_width,3)),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomZoom(0.1),
    ]
)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
```

```
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

Một trong những kĩ thuật khác để giảm hiện tượng overfitting là dùng kĩ thuật Dropout. Khi bạn áp dụng Dropout cho một lớp, nó sẽ loại bỏ ngẫu nhiên một số đơn vị đầu ra từ tầng trong quá trình đào tạo. Dropout lấy một số phân số làm giá trị đầu vào, ở dạng như 0,1, 0,2, 0,4, v.v. Điều này có nghĩa là loại bỏ ngẫu nhiên 10%, 20% hoặc 40% đơn vị đầu ra khỏi vào được áp dụng.

```
model = Sequential([
    data_augmentation,
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 sequential_3 (Sequential)   (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 dropout (Dropout)           (None, 22, 22, 64)        0

 flatten_1 (Flatten)         (None, 30976)             0

 dense_6 (Dense)             (None, 128)               3965056

 dense_7 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

```
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
734/734 [==============================] - 8s 10ms/step - loss: 1.2607 - accuracy: 0.4683 - val_loss: 1.0753 -
Epoch 2/20
734/734 [==============================] - 7s 9ms/step - loss: 1.0204 - accuracy: 0.5967 - val_loss: 0.9337 - v
Epoch 3/20
734/734 [==============================] - 7s 10ms/step - loss: 0.8945 - accuracy: 0.6635 - val_loss: 0.9985 -
Epoch 4/20
734/734 [==============================] - 7s 9ms/step - loss: 0.8290 - accuracy: 0.6771 - val_loss: 0.8367 - v
Epoch 5/20
734/734 [==============================] - 8s 11ms/step - loss: 0.7903 - accuracy: 0.6890 - val_loss: 0.8332 -
Epoch 6/20
734/734 [==============================] - 7s 10ms/step - loss: 0.7726 - accuracy: 0.7071 - val_loss: 0.7944 -
Epoch 7/20
734/734 [==============================] - 7s 9ms/step - loss: 0.7126 - accuracy: 0.7234 - val_loss: 0.7496 - v
Epoch 8/20
734/734 [==============================] - 7s 10ms/step - loss: 0.7009 - accuracy: 0.7282 - val_loss: 0.7853 -
Epoch 9/20
734/734 [==============================] - 7s 9ms/step - loss: 0.6656 - accuracy: 0.7493 - val_loss: 0.7776 - v
Epoch 10/20
734/734 [==============================] - 7s 10ms/step - loss: 0.6338 - accuracy: 0.7585 - val_loss: 0.8121 -
Epoch 11/20
734/734 [==============================] - 7s 10ms/step - loss: 0.6318 - accuracy: 0.7572 - val_loss: 0.7872 -
Epoch 12/20
734/734 [==============================] - 8s 11ms/step - loss: 0.6024 - accuracy: 0.7735 - val_loss: 0.8419 -
Epoch 13/20
734/734 [==============================] - 7s 9ms/step - loss: 0.5814 - accuracy: 0.7762 - val_loss: 0.7535 - v
Epoch 14/20
734/734 [==============================] - 7s 9ms/step - loss: 0.5596 - accuracy: 0.7922 - val_loss: 0.7479 - v
```

```
Epoch 15/20
734/734 [==============================] - 7s 9ms/step - loss: 0.5413 - accuracy: 0.8062 - val_loss: 0.7929 - v
Epoch 16/20
734/734 [==============================] - 7s 9ms/step - loss: 0.5293 - accuracy: 0.7963 - val_loss: 0.7927 - v
Epoch 17/20
734/734 [==============================] - 7s 10ms/step - loss: 0.4997 - accuracy: 0.8096 - val_loss: 0.8218 -
Epoch 18/20
734/734 [==============================] - 7s 10ms/step - loss: 0.4928 - accuracy: 0.8171 - val_loss: 0.8337 -
Epoch 19/20
734/734 [==============================] - 8s 10ms/step - loss: 0.4839 - accuracy: 0.8253 - val_loss: 0.9336 -
Epoch 20/20
734/734 [==============================] - 7s 9ms/step - loss: 0.4491 - accuracy: 0.8386 - val_loss: 0.7777 - v
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Dự đoán với dữ liệu mới

```
sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)
PIL.Image.open(sunflower_path)
```

```
img = keras.preprocessing.image.load_img(
    sunflower_path, target_size=(img_height, img_width)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
1/1 [==============================] - 0s 138ms/step
This image most likely belongs to sunflowers with a 99.80 percent confidence.
```

3. Phân lớp văn bản với RNN

Đối với phân lớp văn bản chúng ta sẽ thực hành với mạng RNN và tập dữ liệu IMDB cho phân tích quan điểm.

3.1. Import thư viện

```
```

```
import numpy as np

import tensorflow_datasets as tfds
import tensorflow as tf

tfds.disable_progress_bar()
```

```
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
    plt.plot(history.history[metric])
    plt.plot(history.history['val_'+metric], '')
    plt.xlabel("Epochs")
    plt.ylabel(metric)
    plt.legend([metric, 'val_'+metric])
```

3.2. Download tập dữ liệu

Bộ dữ liệu đánh giá phim lớn của IMDB là một bộ dữ liệu phân loại nhị phân—tất cả các bài đánh giá đều có cảm xúc tích cực hoặc tiêu cực.

Tải xuống tập dữ liệu bằng TFDS. Xem hướng dẫn tải văn bản để biết chi tiết về cách tải loại dữ liệu này.

```
dataset, info = tfds.load('imdb_reviews', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

train_dataset.element_spec
```

```
(TensorSpec(shape=(), dtype=tf.string, name=None),
 TensorSpec(shape=(), dtype=tf.int64, name=None))
```

```
for example, label in train_dataset.take(1):
    print('text: ', example.numpy())
    print('label: ', label.numpy())
```

```
text:  b"This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Bo
label:  0
```

```
BUFFER_SIZE = 1000
BATCH_SIZE = 64
```

```
train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
```

```
for example, label in train_dataset.take(1):
  print('texts: ', example.numpy()[:3])
  print()
  print('labels: ', label.numpy()[:3])
```

```
texts:  [b'The movie remains in the gray for far too long. Very little gets explained as the movie progresses,
 b'Rosalind Russell executes a power-house performance as Rosie Lord, a very wealthy woman with greedy heirs. W
 b"I have read over 100 of the Nancy Drew books, and if you are not bright enough to catch on yet, Nancy Drew t

labels:  [0 1 1]
```

3.3. Biểu diễn văn bản

Văn bản thô do cần phải được xử lý trước khi có thể sử dụng nó trong một mô hình. Cách đơn giản nhất để xử lý văn bản cho đào tạo là sử dụng tầng experimental.preprocessing.TextVectorization. Ngoài ra có thể tham khảo các cách biểu diễn văn bản khác.

```
VOCAB_SIZE=5000
encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label: text))
```

```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U16')
```

Các văn bản được padding thêm 0 tương ứng với chuỗi dài nhất trong lô.

```
encoded_example = encoder(example)[:3].numpy()
encoded_example
```

```
array([[   2,   18, 1236, ...,    0,    0,    0],
       [   1, 2575,    1, ...,    0,    0,    0],
       [  10,   26,  321, ...,    0,    0,    0]])
```

kí hiệu UNK tương ứng với unknow do chúng ta đã giới hạn kích thước tập từ vựng.

```
for n in range(3):
  print("Original: ", example[n].numpy())
  print("Round-trip: ", " ".join(vocab[encoded_example[n]]))
  print()
```

```
Original:  b'The movie remains in the gray for far too long. Very little gets explained as the movie progresses
Round-trip:  the movie remains in the gray for far too long very little gets explained as the movie progresses

Original:  b'Rosalind Russell executes a power-house performance as Rosie Lord, a very wealthy woman with greed
Round-trip:  [UNK] russell [UNK] a [UNK] performance as [UNK] lord a very wealthy woman with greedy [UNK] with

Original:  b"I have read over 100 of the Nancy Drew books, and if you are not bright enough to catch on yet, Na
Round-trip:  i have read over 100 of the nancy drew books and if you are not bright enough to catch on yet nanc
```

3.4. Mô hình

Định nghĩa kiến trúc mô hình

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

```
print([layer.supports_masking for layer in model.layers])
```

```
[False, True, True, True, True]
```

```
# predict on a sample text without padding.

sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions[0])
```

```
1/1 [==============================] - 3s 3s/step
[0.00325007]
```

```
# predict on a sample text with padding

padding = "the " * 2000
predictions = model.predict(np.array([sample_text, padding]))
print(predictions[0])
```

```
1/1 [==============================] - 0s 120ms/step
[0.00325007]
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (None, 128)              66048
 l)

 dense_8 (Dense)             (None, 64)                8256

 dense_9 (Dense)             (None, 1)                 65

=================================================================
Total params: 394,369
Trainable params: 394,369
Non-trainable params: 0
_________________________________________________________________
```

Training

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/10
391/391 [==============================] - 45s 90ms/step - loss: 0.6195 - accuracy: 0.5929 - val_loss: 0.4302 -
Epoch 2/10
391/391 [==============================] - 31s 80ms/step - loss: 0.3235 - accuracy: 0.8600 - val_loss: 0.3135 -
Epoch 3/10
391/391 [==============================] - 33s 84ms/step - loss: 0.2490 - accuracy: 0.8968 - val_loss: 0.2885 -
Epoch 4/10
391/391 [==============================] - 31s 81ms/step - loss: 0.2161 - accuracy: 0.9129 - val_loss: 0.3023 -
Epoch 5/10
391/391 [==============================] - 32s 81ms/step - loss: 0.1941 - accuracy: 0.9232 - val_loss: 0.3132 -
Epoch 6/10
391/391 [==============================] - 33s 84ms/step - loss: 0.1826 - accuracy: 0.9288 - val_loss: 0.3127 -
Epoch 7/10
391/391 [==============================] - 33s 84ms/step - loss: 0.1700 - accuracy: 0.9354 - val_loss: 0.3215 -
Epoch 8/10
391/391 [==============================] - 33s 85ms/step - loss: 0.1604 - accuracy: 0.9384 - val_loss: 0.3475 -
Epoch 9/10
391/391 [==============================] - 32s 81ms/step - loss: 0.1523 - accuracy: 0.9434 - val_loss: 0.3956 -
Epoch 10/10
391/391 [==============================] - 33s 85ms/step - loss: 0.1455 - accuracy: 0.9462 - val_loss: 0.4018 -
```

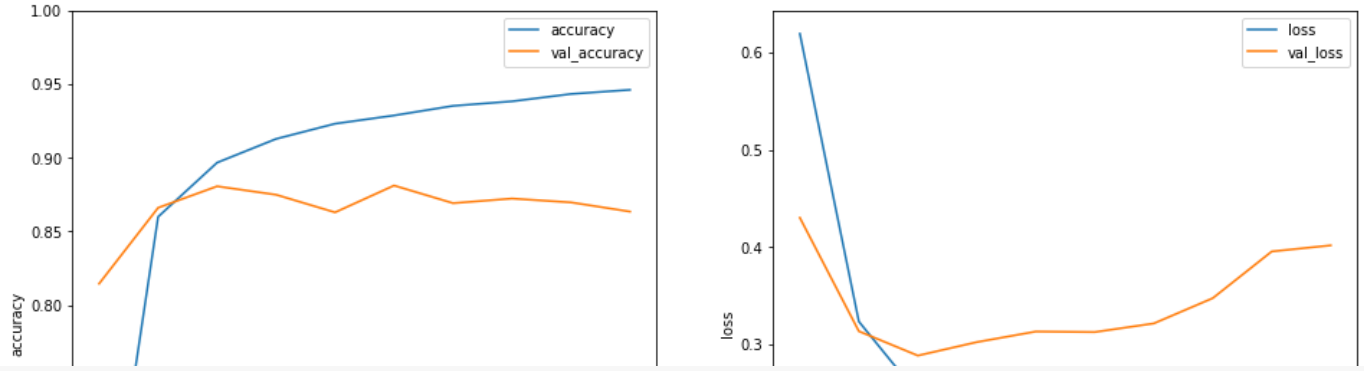
3.5 Đánh giá mô hình

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================] - 18s 46ms/step - loss: 0.3876 - accuracy: 0.8605
Test Loss: 0.3876105546951294
Test Accuracy: 0.860480010509491
```

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plot_graphs(history, 'accuracy')
plt.ylim(None,1)
plt.subplot(1,2,2)
plot_graphs(history, 'loss')
plt.ylim(0,None)
```

```
(0.0, 0.6431896567344666)
```




```
sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
```

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

3.6. Cải tiến mô hình

Sử dụng nhiều tầng LSTM

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding_1 (Embedding)     (None, None, 64)          320000

 bidirectional_1 (Bidirectio  (None, None, 128)        66048
 nal)

 bidirectional_2 (Bidirectio  (None, 64)               41216
 nal)

 dense_10 (Dense)            (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_11 (Dense)            (None, 1)                 65
```

```
=================================================================
Total params: 431,489
Trainable params: 431,489
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

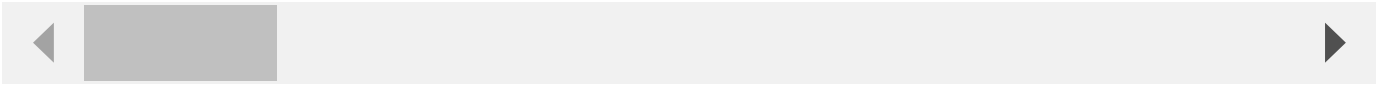
```
Epoch 1/10
391/391 [==============================] - 86s 155ms/step - loss: 0.5980 - accuracy: 0.6123 - val_loss: 0.3725
Epoch 2/10
391/391 [==============================] - 59s 152ms/step - loss: 0.3048 - accuracy: 0.8748 - val_loss: 0.3108
Epoch 3/10
391/391 [==============================] - 56s 143ms/step - loss: 0.2381 - accuracy: 0.9079 - val_loss: 0.2994
Epoch 4/10
391/391 [==============================] - 56s 143ms/step - loss: 0.2069 - accuracy: 0.9223 - val_loss: 0.3421
Epoch 5/10
391/391 [==============================] - 56s 144ms/step - loss: 0.1986 - accuracy: 0.9273 - val_loss: 0.3401
Epoch 6/10
391/391 [==============================] - 57s 147ms/step - loss: 0.1721 - accuracy: 0.9387 - val_loss: 0.3410
Epoch 7/10
391/391 [==============================] - 71s 181ms/step - loss: 0.1529 - accuracy: 0.9471 - val_loss: 0.3976
Epoch 8/10
391/391 [==============================] - 56s 143ms/step - loss: 0.1434 - accuracy: 0.9512 - val_loss: 0.4056
Epoch 9/10
391/391 [==============================] - 58s 148ms/step - loss: 0.1354 - accuracy: 0.9550 - val_loss: 0.4596
Epoch 10/10
391/391 [==============================] - 55s 140ms/step - loss: 0.1214 - accuracy: 0.9607 - val_loss: 0.4615
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================] - 29s 75ms/step - loss: 0.4276 - accuracy: 0.8715
Test Loss: 0.4276154339313507
Test Accuracy: 0.8714799880981445
```

```
# predict on a sample text without padding.

sample_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions)
```

```
WARNING:tensorflow:5 out of the last 318 calls to <function Model.make_predict_function.<locals>.predict_functi
1/1 [==============================] - 6s 6s/step
[[-2.342837]]
```



```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plot_graphs(history, 'accuracy')
plt.subplot(1,2,2)
plot_graphs(history, 'loss')
```