

# Deep Learning trong khoa học dữ liệu

## Bài tập thực hành 1 - DS201.M11.2

Đỗ Trọng Hợp, Lưu Thanh Sơn, Nguyễn Thành Luân  
Sinh viên: Phạm Đức Thế - 19522253  
Ngôn ngữ lập trình: Python

Thứ 4, ngày 29 tháng 09 năm 2021

## Bài tập: XÂY DỰNG MẠNG NEURAL ĐƠN GIẢN

### Hướng dẫn thực hành

#### 1. Import các thư viện cần thiết

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sn

from keras.datasets.mnist import load_data
from tensorflow.keras.utils import to_categorical
from keras.layers import Dense, Activation
from keras.models import Sequential
from sklearn.metrics import accuracy_score, confusion_matrix
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
from keras.models import load_model
```

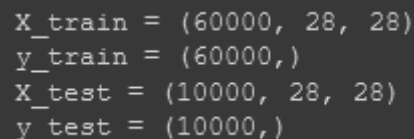
#### 2. Load bộ dữ liệu MNIST

```
(X_train, y_train), (X_test, y_test) = load_data()
```

#### 3. Chuẩn bị dữ liệu

```
print('X_train = {}'.format(X_train.shape))
print('y_train = {}'.format(y_train.shape))
print('X_test = {}'.format(X_test.shape))
print('y_test = {}'.format(y_test.shape))
```

– Output: Hình 1



```
X_train = (60000, 28, 28)
y_train = (60000,)
X_test = (10000, 28, 28)
y_test = (10000,)
```

Hình 1: Shape của dữ liệu

– X\_train gồm có 60,000 bức ảnh trắng đen (channel = 1) về chữ số viết tay với kích thước mỗi bức ảnh là  $28 \times 28$  pixel dùng để train model, y\_train là các nhãn tương ứng X\_train.

- X\_test gồm có 10,000 bức ảnh trắng đen (channel = 1) về chữ số viết tay với kích thước mỗi bức ảnh là  $28 \times 28$  pixel dùng để test model, y\_test là các nhãn tương ứng của X\_test.

---

```
# Reshape kích thước của dữ liệu
X_train_reshaped = X_train.reshape(-1, 28*28)
X_test_reshaped = X_test.reshape(-1, 28*28)
```

---

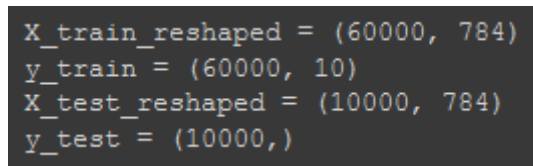
```
y_train = to_categorical(y_train, num_classes = 10)
```

---

```
# Xem kích thước của dữ liệu sau khi reshape
print('X_train_reshaped = {}'.format(X_train_reshaped.shape))
print('y_train = {}'.format(y_train.shape))
print('X_test_reshaped = {}'.format(X_test_reshaped.shape))
print('y_test = {}'.format(y_test.shape))
```

---

- Output: Hình 2



```
X_train_reshaped = (60000, 784)
y_train = (60000, 10)
X_test_reshaped = (10000, 784)
y_test = (10000,)
```

Hình 2: Shape của dữ liệu sau khi reshape

#### 4. Xây dựng mạng neural bằng Keras

---

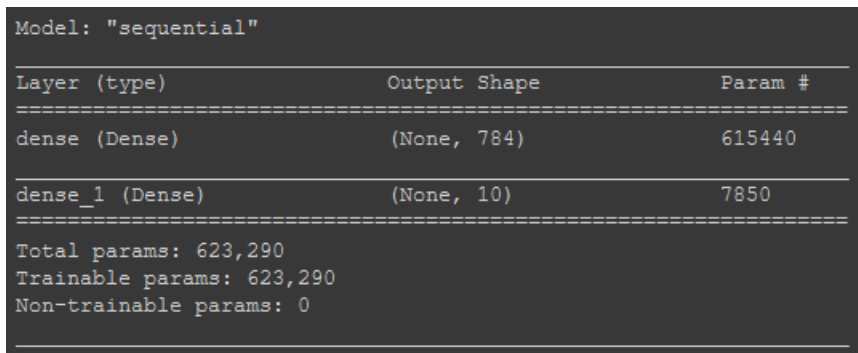
```
# Khởi tạo model
model = Sequential()
```

```
model.add(Dense(784, input_shape=(784,)))
model.add(Dense(10, input_shape=(10,)))
```

```
model.summary()
```

---

- Output: Hình 3



```
Model: "sequential"
Layer (type)                Output Shape              Param #
=====
dense (Dense)                (None, 784)               615440
dense_1 (Dense)              (None, 10)                7850
=====
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
```

Hình 3: Summary model

#### \* Nhận xét:

- Model có 1 hidden layer Dense, có shape là (None, 784) trong đó : None sẽ được xác định trong quá trình đưa dữ liệu vào training model (tổng số điểm dữ liệu training), 784 là shape dữ liệu đầu vào của model.
- Tổng số params mà model cần phải học là 623,290 params.

## 5. Huấn luyện mô hình

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = BinaryCrossentropy()
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
model.fit(X_train_reshaped, y_train, batch_size=128,
          epochs=10, validation_split=0.2)
```

## 6. Đánh giá mô hình

```
y_pred = model.predict(X_test_reshaped)
y_pred = np.argmax(y_pred, axis=-1)
accuracy = round(accuracy_score(y_test, y_pred)*100, 2)
print('Accuracy test = {}'.format(accuracy))
# Output: Accuracy test = 9.95%
```

### \* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test rất thấp (Accuracy test = 9.95%).
- Nguyên nhân là vì mô hình chưa sử dụng các hàm kích hoạt (activation function), nên mô hình chưa rút trích được các đặc trưng quan trọng trong quá trình training, nên mô hình có accuracy thấp.

## 7. Lưu mô hình

```
# Lưu model qua drive
model.save('/content/drive/MyDrive/my_model.h5')
# Load model
model = load_model('/content/drive/MyDrive/my_model.h5')
```

## Bài 1: Hãy sử dụng activation sigmoid và xem kết quả.

- Các thư viện cần thiết và dữ liệu sử dụng lại từ phần [Hướng dẫn thực hành](#).

### 1. Xây dựng mạng neural bằng Keras

```
# Khởi tạo model
model_1 = Sequential()

model_1.add(Dense(784, input_shape=(784, ), activation='sigmoid'))
model_1.add(Dense(10, input_shape=(10, ), activation='sigmoid'))

model_1.summary()
```

- Output: Hình 4

```
Model: "sequential_1"
_____
Layer (type)                 Output Shape              Param #
=====
dense_2 (Dense)              (None, 784)               615440
_____
dense_3 (Dense)              (None, 10)                7850
=====
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
_____
```

Hình 4: Summary model bài 1

\* **Nhận xét:**

- Model có 1 hidden layer Dense, có shape là (None, 784) trong đó : None sẽ được xác định trong quá trình đưa dữ liệu vào training model (tổng số điểm dữ liệu training), 784 là shape dữ liệu đầu vào của model.
- Tổng số params mà model cần phải học là 623,290 params.

2. Huấn luyện mô hình

---

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = BinaryCrossentropy()
model_1.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
model_1.fit(X_train_reshaped, y_train, batch_size=128,
            epochs=20, validation_split=0.2)
```

---

3. Đánh giá mô hình

---

```
y_pred_1 = model_1.predict(X_test_reshaped)
y_pred_1 = np.argmax(y_pred_1, axis = -1)
accuracy_1 = round(accuracy_score(y_test, y_pred_1)*100,2)
print('Accuracy test = {}'.format(accuracy_1))
# Output: Accuracy test = 88.62%
```

---

\* **Nhận xét:**

- Accuracy của mô hình khi kiểm tra trên tập test cao (Accuracy test = 88.62%).
- Nguyên nhân là vì mô hình đã có sử dụng các hàm kích hoạt (activation function) cụ thể activation='sigmoid', nên mô hình có thể rút trích được các đặc trưng quan trọng trong quá trình training, nên mô hình có accuracy cao.

## Bài 2: Hãy sử dụng activation relu cho lớp thứ 1 và activation sigmoid cho lớp đầu ra và xem kết quả.

- Các thư viện cần thiết và dữ liệu sử dụng lại từ phần [Hướng dẫn thực hành](#).

1. Xây dựng mạng neural bằng Keras

---

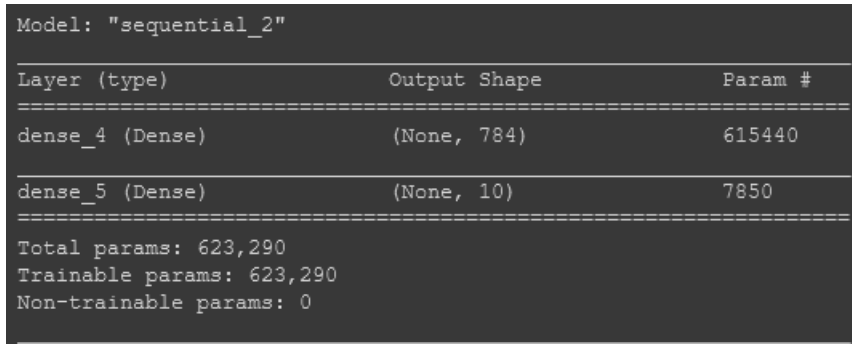
```
# Khởi tạo model
model_2 = Sequential()

model_2.add(Dense(784, input_shape=(784, ), activation='relu'))
model_2.add(Dense(10, input_shape=(10, ), activation='sigmoid'))

model_2.summary()
```

---

- Output: Hình 5



```
Model: "sequential_2"
Layer (type)                 Output Shape              Param #
=====
dense_4 (Dense)              (None, 784)              615440
dense_5 (Dense)              (None, 10)              7850
=====
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
```

Hình 5: Summary model bài 2

\* **Nhận xét:**

- Model có 1 hidden layer Dense, có shape là (None, 784) trong đó : None sẽ được xác định trong quá trình đưa dữ liệu vào training model (tổng số điểm dữ liệu training), 784 là shape dữ liệu đầu vào của model.
- Tổng số params mà model cần phải học là 623,290 params.

2. Huấn luyện mô hình

---

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = BinaryCrossentropy()
model_2.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
model_2.fit(X_train_reshaped, y_train, batch_size=128,
            epochs=20, validation_split=0.2)
```

---

3. Đánh giá mô hình

---

```
y_pred_2 = model_2.predict(X_test_reshaped)
y_pred_2 = np.argmax(y_pred_2, axis = -1)
accuracy_2 = round(accuracy_score(y_test, y_pred_2)*100,2)
print('Accuracy test = {}'.format(accuracy_2))
# Output: Accuracy test = 74.68%
```

---

\* **Nhận xét:**

- Accuracy của mô hình khi kiểm tra trên tập test khá cao (Accuracy test = 74.68%).
- Nguyên nhân là vì mô hình đã có sử dụng các hàm kích hoạt (activation function) cụ thể activation='relu' ở lớp thứ 1 và activation='sigmoid' ở lớp đầu ra, nên mô hình có thể rút trích được các đặc trưng quan trọng trong quá trình training, nên mô hình có accuracy khá cao.
- Accuracy của mô hình này thấp hơn accuracy của mô hình ở bài 1 nên có thể nhận xét là activation='relu' ở lớp thứ 1 kém hiệu quả hơn khi mà activation='sigmoid' ở lớp thứ 1. Có nghĩa là khi activation='sigmoid' ở lớp thứ 1 sẽ rút trích được nhiều thông tin quan trọng trong quá trình training hơn activation='relu'.

### Bài 3: Normalizing giá trị cho data như sau:

---

```
X_train_new = X_train/255.0
X_test_new = X_test/255.0
```

---

Hãy thực hiện lại model ở bài 2 trên dữ liệu X\_train và X\_test đã chuẩn hoá.

- Các thư viện cần thiết và load dữ liệu sử dụng lại từ phần [Hướng dẫn thực hành](#).

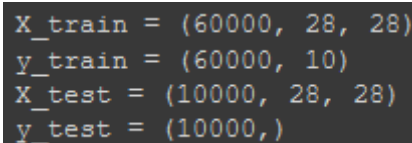
1. Chuẩn bị dữ liệu

---

```
print('X_train = {}'.format(X_train.shape))
print('y_train = {}'.format(y_train.shape))
print('X_test = {}'.format(X_test.shape))
print('y_test = {}'.format(y_test.shape))
```

---

- Output: Hình 6



```
X_train = (60000, 28, 28)
y_train = (60000, 10)
X_test = (10000, 28, 28)
y_test = (10000,)
```

Hình 6: Shape của dữ liệu

- X\_train gồm có 60,000 bức ảnh trắng đen (channel = 1) về chữ số viết tay với kích thước mỗi bức ảnh là  $28 \times 28$  pixel dùng để train model, y\_train là các nhãn tương ứng X\_train.
- X\_test gồm có 10,000 bức ảnh trắng đen (channel = 1) về chữ số viết tay với kích thước mỗi bức ảnh là  $28 \times 28$  pixel dùng để test model, y\_test là các nhãn tương ứng của X\_test.

---

```
# Data normalization
X_train_new = X_train/255.0
X_test_new = X_test/255.0

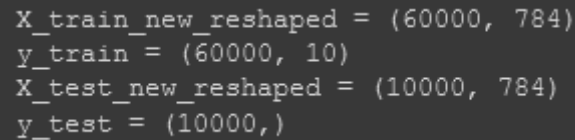
# Reshape kích thước của dữ liệu
X_train_new_reshaped = X_train_new.reshape(-1, 28*28)
X_test_new_reshaped = X_test_new.reshape(-1, 28*28)
```

---

```
# Xem kích thước của dữ liệu sau khi reshape
print('X_train_new_reshaped = {}'.format(X_train_new_reshaped.shape))
print('y_train = {}'.format(y_train.shape))
print('X_test_new_reshaped = {}'.format(X_test_new_reshaped.shape))
print('y_test = {}'.format(y_test.shape))
```

---

- Output: Hình 7



```
X_train_new_reshaped = (60000, 784)
y_train = (60000, 10)
X_test_new_reshaped = (10000, 784)
y_test = (10000,)
```

Hình 7: Shape của dữ liệu sau khi reshape

## 2. Xây dựng mạng neural bằng Keras

---

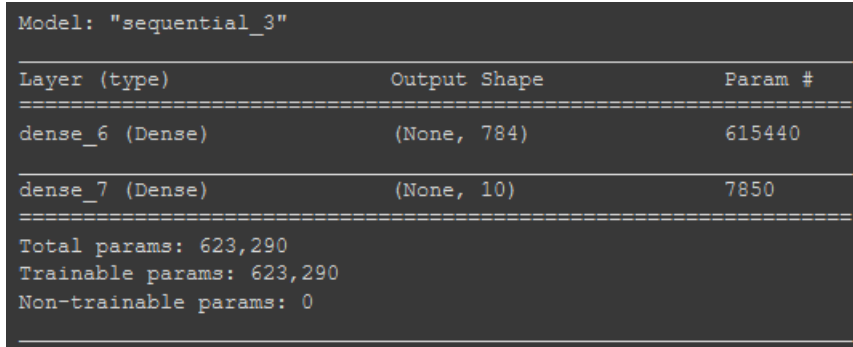
```
# Khởi tạo model
model_3 = Sequential()

model_3.add(Dense(784, input_shape=(784, ), activation='relu'))
model_3.add(Dense(10, input_shape=(10, ), activation='sigmoid'))

model_3.summary()
```

---

- Output: Hình 8



```
Model: "sequential_3"
Layer (type)                Output Shape              Param #
=====
dense_6 (Dense)              (None, 784)               615440
dense_7 (Dense)              (None, 10)                7850
=====
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
```

Hình 8: Summary model bài 3

### \* Nhận xét:

- Model có 1 hidden layer Dense, có shape là (None, 784) trong đó : None sẽ được xác định trong quá trình đưa dữ liệu vào training model (tổng số điểm dữ liệu training), 784 là shape dữ liệu đầu vào của model.
- Tổng số params mà model cần phải học là 623,290 params.

### 3. Huấn luyện mô hình

---

```
# Compile model
optimizer = Adam(learning_rate=0.01)
loss = BinaryCrossentropy()
model_3.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
model_3.fit(X_train_new_resaped, y_train, batch_size=128,
            epochs=20, validation_split=0.2)
```

---

### 4. Đánh giá mô hình

---

```
y_pred_3 = model_3.predict(X_test_new_resaped)
y_pred_3 = np.argmax(y_pred_3, axis = -1)
accuracy_3 = round(accuracy_score(y_test, y_pred_3)*100,2)
print('Accuracy test = {}'.format(accuracy_3))
# Output: Accuracy test = 97.98%
```

---

#### \* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test rất cao (Accuracy test = 97.98%), cao hơn so với mô hình ở bài 2.
- Nguyên nhân là vì dữ liệu đầu vào đã được chuẩn hoá, các pixel của dữ liệu nằm trong khoảng từ 0–255 (0–đen, 255–trắng), ta chuẩn hoá bằng cách chia X\_train, X\_test cho 255.0 ta sẽ chuẩn hoá các pixel của các bức ảnh về trong khoảng từ 0–1. Khi đó mô hình sẽ học được hiệu hơn. Nên mô hình có accuracy rất cao.

### Bài 4: In ra ma trận nhầm lẫn của mô hình ở bài 3 và nhận xét.

---

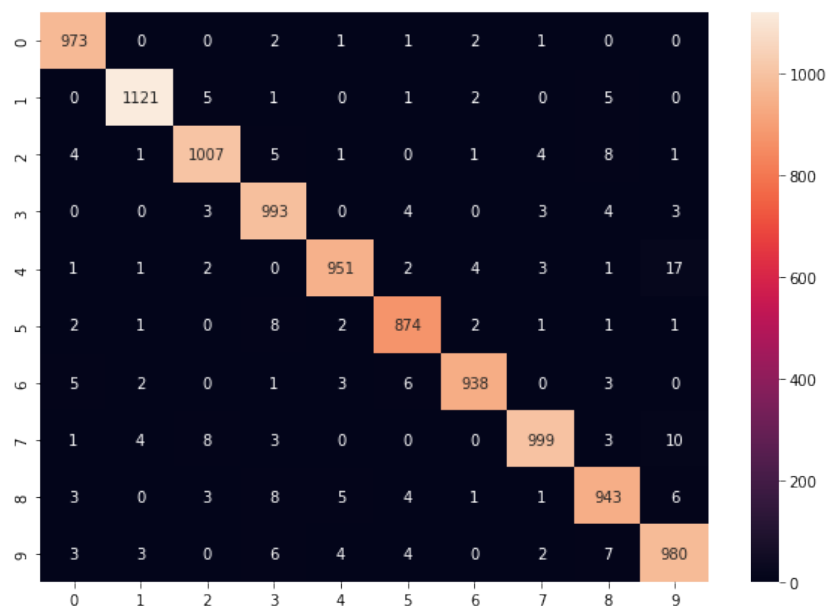
```
cm = confusion_matrix(y_test, y_pred_3)

df_cm = pd.DataFrame(cm, index = [i for i in range(0,10)],
                     columns = [i for i in range(0,10)])

plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='.5g')
```

---

– Output: Hình 9



Hình 9: Ma trận nhầm lẫn của bài 3

\* **Nhận xét:**

- Mô hình dự đoán chính xác nhất ở class 1 với 973/980 (chiếm 99.29%) điểm dữ liệu được dự đoán chính xác, và cũng là class có số lượng dự đoán nhầm ít nhất với 7 điểm dữ liệu bị dự đoán nhầm.
- Mô hình dự đoán tệ nhất ở class 4 với 951/982 (chiếm 96.84%) điểm dữ liệu được dự đoán chính xác, và cũng là một trong 2 class có số lượng dự đoán nhầm nhiều nhất với 31 điểm dữ liệu bị dự đoán nhầm (class 8 cũng có số lượng điểm dữ liệu bị dự đoán nhầm là 31), trong đó class 4 bị dự đoán nhầm là class 9 là nhiều nhất với 17 điểm dữ liệu bị dự đoán nhầm.
- Nhìn chung kết quả của mô hình thu được vẫn rất tốt, độ chính xác của từng class từ 96.84% đến 99.29%.

## Bài 5: Hãy xây dựng model đơn giản gồm 2 lớp cho bộ dữ liệu small CIFAR10.

### 1. Import các thư viện cần thiết

- Các thư viện cần thiết sử dụng lại từ phần [Hướng dẫn thực hành](#).

---

```
from keras.datasets.cifar10 import load_data
```

---

### 2. Load bộ dữ liệu CIFAR10

---

```
(X_train_5, y_train_5), (X_test_5, y_test_5) = load_data()
```

---

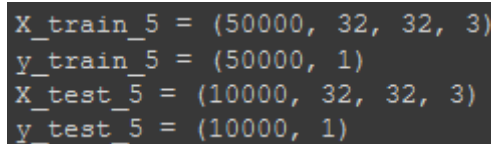
### 3. Chuẩn bị dữ liệu

---

```
print('X_train_5 = {}'.format(X_train_5.shape))
print('y_train_5 = {}'.format(y_train_5.shape))
print('X_test_5 = {}'.format(X_test_5.shape))
print('y_test_5 = {}'.format(y_test_5.shape))
```

---

- Output: Hình 10



```
X_train_5 = (50000, 32, 32, 3)
y_train_5 = (50000, 1)
X_test_5 = (10000, 32, 32, 3)
y_test_5 = (10000, 1)
```

Hình 10: Shape của dữ liệu

- X\_train\_5 gồm có 50,000 bức ảnh màu (số channel = 3) với kích thước mỗi bức ảnh màu là  $32 \times 32$  pixel dùng để train model, y\_train\_5 là các nhãn tương ứng X\_train\_5.

- X\_test\_5 gồm có 10,000 bức ảnh màu (số channel = 3) với kích thước mỗi bức ảnh màu là  $32 \times 32$  pixel dùng để test model, y\_test\_5 là các nhãn tương ứng của X\_test\_5.

---

```
# Data normalization
```

```
X_train_5_new = X_train_5/255.0
```

```
X_test_5_new = X_test_5/255.0
```

---

```
# Reshape kích thước của dữ liệu
```

```
X_train_5_resaped = X_train_5_new.reshape(-1, 32*32*3)
```

```
X_test_5_resaped = X_test_5_new.reshape(-1, 32*32*3)
```

---

```
y_train_5 = to_categorical(y_train_5, num_classes = 10)
```

---

---

```
# Xem kích thước của dữ liệu sau khi reshape
```

```
print('X_train_5_resaped = {}'.format(X_train_5_resaped.shape))
```

```
print('y_train_5 = {}'.format(y_train_5.shape))
```

```
print('X_test_5_resaped = {}'.format(X_test_5_resaped.shape))
```

```
print('y_test_5 = {}'.format(y_test_5.shape))
```

---



– Output: Hình 11

```
X_train_5_resaped = (50000, 3072)
y_train_5 = (50000, 10)
X_test_5_resaped = (10000, 3072)
y_test_5 = (10000, 1)
```

Hình 11: Shape của dữ liệu sau khi reshape

#### 4. Xây dựng mạng neural bằng Keras

---

```
# Khởi tạo model
model_5 = Sequential()

model_5.add(Dense(3072, input_shape=(3072, ), activation='relu'))
model_5.add(Dense(512, input_shape=(512, ), activation='relu'))
model_5.add(Dense(10, input_shape=(10, ), activation='sigmoid'))

model_5.summary()
```

---

– Output: Hình 12

```
Model: "sequential_4"
_____
Layer (type)                 Output Shape              Param #
=====
dense_8 (Dense)              (None, 3072)             9440256
_____
dense_9 (Dense)              (None, 512)              1573376
_____
dense_10 (Dense)             (None, 10)               5130
=====
Total params: 11,018,762
Trainable params: 11,018,762
Non-trainable params: 0
_____
```

Hình 12: Summary model

#### \* Nhận xét:

- Model có 2 hidden layer Dense.
- Dense thứ 1 có shape là (None, 3072) trong đó : None sẽ được xác định trong quá trình đưa dữ liệu vào training model, 3072 là shape đầu vào của model.
- Dense thứ 2 có shape là (None, 512) trong đó : None sẽ được xác định trong quá trình đưa dữ liệu vào training model, 512 là shape đầu vào của layer Dense thứ 2.
- Tổng số params mà model cần phải học là 11,018,762 params.

#### 5. Huấn luyện mô hình

---

```
# Compile model
optimizer = Adam(learning_rate=0.0001)
loss = BinaryCrossentropy()
model_5.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Training model
history = model_5.fit(X_train_5_resaped, y_train_5, batch_size=128,
                      epochs=30, validation_split=0.2)
```

---

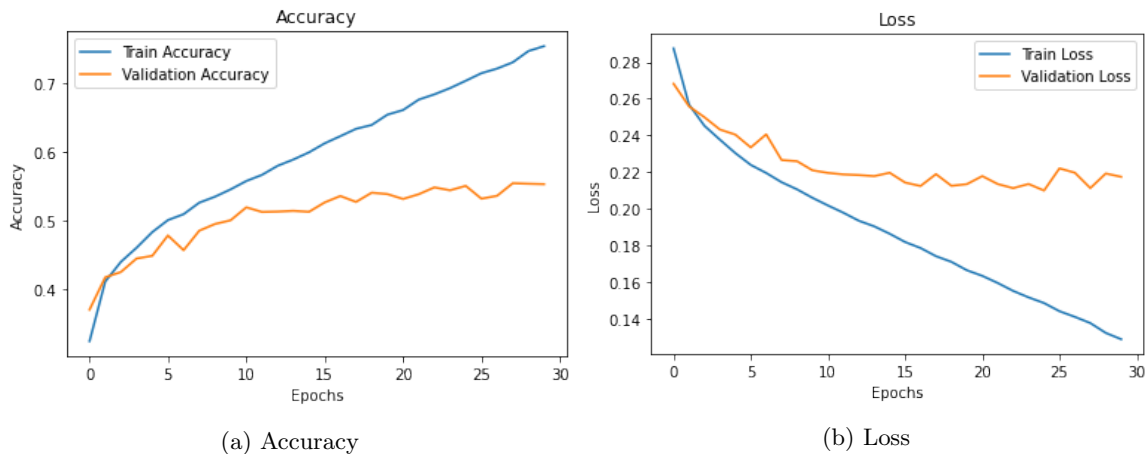
## 6. Đánh giá mô hình

```
plt.figure(0)
plt.plot(history.history['accuracy'], label = 'Train Accuracy')
plt.plot(history.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 13a

```
plt.figure(0)
plt.plot(history.history['loss'], label = 'Train Loss')
plt.plot(history.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 13b



Hình 13: Accuracy và Loss của model bài 5

### \* Nhận xét:

- Accuracy trên tập train vẫn còn tăng nhưng accuracy trên tập validation đã bắt đầu hội tụ từ epoch thứ 20.
- Loss trên tập train vẫn còn giảm nhưng loss trên tập validation bắt đầu hội tụ từ epoch thứ 15.

```
y_pred_5 = model_5.predict(X_test_5_resaped)
y_pred_5 = np.argmax(y_pred_5, axis = -1)
accuracy_5 = round(accuracy_score(y_test_5, y_pred_5)*100,2)
print('Accuracy test = {}'.format(accuracy_5))
# Output: Accuracy test = 55.1%
```

### \* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test khá thấp (Accuracy test = 55.1%).
- Nguyên nhân có thể là vì đây là một mô hình đơn giản, có ít layers, chưa học được các đặc trưng quan trọng, dữ liệu phức tạp cần một mô hình phức tạp hơn mới có thể cho được kết quả tốt. Nên accuracy của mô hình khá thấp.

## 7. Ma trận nhầm lẫn

---

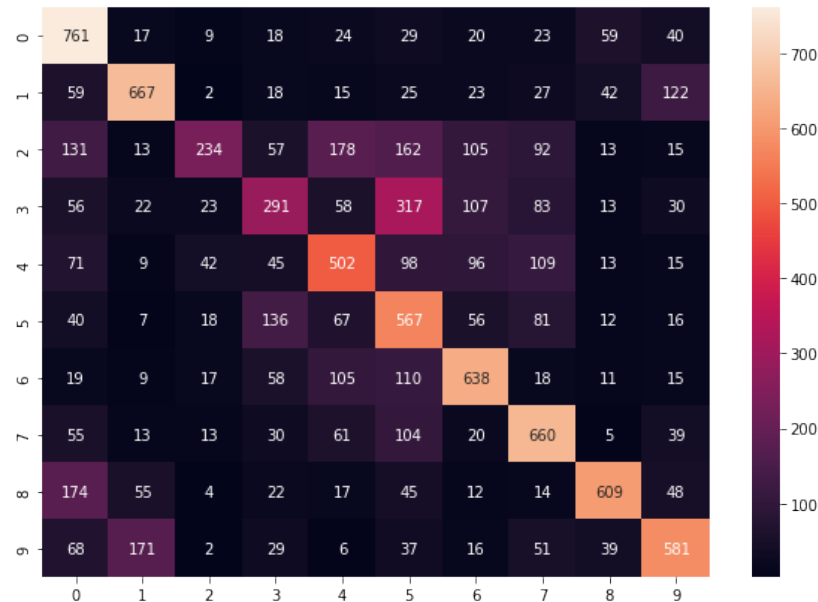
```
cm_5 = confusion_matrix(y_test_5, y_pred_5)

df_cm_5 = pd.DataFrame(cm_5, index = [i for i in range(0,10)],
                        columns = [i for i in range(0,10)])

plt.figure(figsize = (10,7))
sn.heatmap(df_cm_5, annot=True, fmt='.5g')
```

---

– Output: Hình 14



Hình 14: Ma trận nhầm lẫn của bài 5

### \* Nhận xét:

- Mô hình dự đoán tốt ở class 0 với 761/1000 (chiếm 76.1%) điểm dữ liệu được dự đoán chính xác, và cũng là class có số lượng dự đoán nhầm ít nhất với 239 điểm dữ liệu bị dự đoán nhầm. Trong đó class 0 bị dự đoán nhầm là class 8 là nhiều nhất với 59 điểm dữ liệu bị dự đoán nhầm.
- Mô hình dự đoán tệ nhất ở class 2 với 234/1000 (chiếm 23.4%) điểm dữ liệu được dự đoán chính xác, và cũng là class có số lượng dự đoán nhầm nhiều nhất với 766 điểm dữ liệu bị dự đoán nhầm. Trong đó class 2 bị dự đoán nhầm là class 4 là nhiều nhất với 178 điểm dữ liệu bị dự đoán nhầm.
- Class 3 cũng dự đoán rất tệ với 291/1000 (chiếm 29.1%) điểm dữ liệu được dự đoán chính xác, đặc biệt class 3 bị dự đoán nhầm là class 5 với 317 điểm dữ liệu bị dự đoán nhầm và cũng là số điểm dữ liệu bị dự đoán nhầm cao nhất của mô hình đối với tập dữ liệu.
- Nhìn chung đối với 1 mô hình đơn giản chỉ có 2 lớp dense thì mô hình vẫn học được các đặc trưng và cho kết quả tương đối.

## 8. Lưu mô hình

---

```
# Lưu model qua drive
model_5.save('/content/drive/MyDrive/my_model_5.h5')
# Load model
model_5 = load_model('/content/drive/MyDrive/my_model_5.h5')
```

---