

Deep Learning trong khoa học dữ liệu

Bài tập thực hành 3 - 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 27 tháng 10 năm 2021

Bài tập: MẠNG NEURAL TÍCH CHẬP

Set up

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 tensorflow.keras import layers
from keras.layers import Dense, Activation, Input, MaxPooling2D,
from keras.layers import Conv2D, Flatten, Dropout, AveragePooling2D
from keras.models import Sequential
from sklearn.metrics import accuracy_score, confusion_matrix
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.losses import CategoricalCrossentropy
from keras.models import load_model
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping

sn.set() # Set theme
```

2. Load bộ dữ liệu MNIST

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

3. Chuẩn bị dữ liệu

```
X_train, X_dev, y_train, y_dev = train_test_split(X_train, y_train,
                                                    test_size=0.1)

print('X_train = {}'.format(X_train.shape))
# Output: X_train = (54000, 28, 28)
print('y_train = {}'.format(y_train.shape))
# Output: y_train = (54000,)
print('X_dev = {}'.format(X_dev.shape))
# Output: X_dev = (6000, 28, 28)
print('y_dev = {}'.format(y_dev.shape))
# Output: y_dev = (6000,)
print('X_test = {}'.format(X_test.shape))
# Output: X_test = (10000, 28, 28)
print('y_test = {}'.format(y_test.shape))
# Output: y_test = (10000,)
```

- X_train gồm có 60,000 bức ảnh trắng đen (channel = 1) về chữ số viết tay với kích thước mỗi bức ảnh là 28×28 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 28×28 pixel dùng để test model, y_test là các nhãn tương ứng của X_test.

```
# Mo rong them 1 chieu du lieu
X_train_expanded = np.expand_dims(X_train, axis=3)
X_dev_expanded = np.expand_dims(X_dev, axis=3)
X_test_expanded = np.expand_dims(X_test, axis=3)

y_train_new = to_categorical(y_train, num_classes = 10)
y_dev_new = to_categorical(y_dev, num_classes = 10)

print('X_train_expanded = {}'.format(X_train_expanded.shape))
# Output: X_train_expanded = (54000, 28, 28, 1)
print('y_train_new = {}'.format(y_train_new.shape))
# Output: y_train_new = (54000, 10)
print('X_dev_expanded = {}'.format(X_dev_expanded.shape))
# Output: X_dev_expanded = (6000, 28, 28, 1)
print('y_dev_new = {}'.format(y_dev_new.shape))
# Output: y_dev_new = (6000, 10)
print('X_test_expanded = {}'.format(X_test_expanded.shape))
# Output: X_test_expanded = (10000, 28, 28, 1)
print('y_test = {}'.format(y_test.shape))
# Output: y_test = (10000,)
```

Bài 1: Hiện thực mô hình ở phần 3, sử dụng hàm activation relu cho 2 lớp CONV. Thực hiện huấn luyện lại mô hình và xem kết quả? Cho biết độ chính xác và đồ thị học của mô hình.

1. Xây dựng mạng neural bằng Keras

```
# Khoi tao model
model_1 = Sequential()

model_1.add(Input(shape=(28, 28, 1)))
model_1.add(Conv2D(32, padding='valid', kernel_size=(3, 3),
                    activation='relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))

model_1.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))

model_1.add(Flatten())
model_1.add(Dense(10, activation='softmax'))

model_1.summary()
```

- Output: Hình 1

* **Nhận xét:**

- Model có Input shape là (28,28,1)
- Model có 2 layers *Conv2D*. Layer thứ nhất có 32 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*. Layer thứ hai có 64 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*.
- Model có 2 layers *MaxPooling* có cùng kích thước là 2×2 .
- Tổng số params mà model cần phải học là 34,826 params.

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 10)	16010
=====		
Total params: 34,826		
Trainable params: 34,826		
Non-trainable params: 0		

Hình 1: Summary model

2. Huấn luyện mô hình

```
# Compile model
Optimizer = Adam(learning_rate=1e-3)
Loss = CategoricalCrossentropy()
model_1.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_1 = model_1.fit(X_train_expanded, y_train_new,
                        validation_data=(X_dev_expanded, y_dev_new),
                        batch_size=128, epochs=30)
```

3. Visualization Loss & Accuracy training

```
plt.plot(history_1.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_1.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 2a

```
plt.plot(history_1.history['loss'], label = 'Train Loss')
plt.plot(history_1.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 2b

* Nhận xét:

- Accuracy trên tập train và tập dev đều rất cao (tập train: $\sim 99.7\%$, tập dev: $\sim 98.4\%$).
- Loss trên cả 2 tập train và dev đều giảm rất nhanh và hội tụ rất sớm (khoảng epoch thứ 5) nhưng loss trên tập dev có hiện tượng tăng lên ở những epoch cuối, cho thấy mô hình bắt đầu xuất hiện tình trạng overfitting.



Hình 2: Accuracy và Loss của model bài 1

4. Đánh giá mô hình

```

y_pred = model_1.predict(X_test_expanded)
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 = 98.76%

```

* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test cao (Accuracy test = 98.76%). Có thể thấy đây là một mô hình rất tốt.

Bài 2: Hiện thực mô hình ở phần 3, sử dụng hàm activation tanh cho 2 lớp CONV. Thực hiện huấn luyện lại mô hình và xem kết quả? Cho biết độ chính xác và đồ thị học của mô hình.

1. Xây dựng mạng neural bằng Keras

```

# Khởi tạo model
model_2 = Sequential()

model_2.add(Input(shape=(28, 28, 1)))
model_2.add(Conv2D(32, padding='valid', kernel_size=(3, 3),
                    activation='tanh'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

model_2.add(Conv2D(64, kernel_size=(3, 3), activation='tanh'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

model_2.add(Flatten())
model_2.add(Dense(10, activation='softmax'))

model_2.summary()

```

– Output: Hình 3

* Nhận xét:

- Model có Input shape là (28,28,1)
- Model có 2 layers *Conv2D*. Layer thứ nhất có 32 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *tanh*. Layer thứ hai có 64 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *tanh*.
- Model có 2 layers *MaxPooling* có cùng kích thước là 2×2 .
- Tổng số params mà model cần phải học là 34,826 params.

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_1 (Dense)	(None, 10)	16010

```

Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0

```

Hình 3: Summary model

2. Huấn luyện mô hình

```

# Compile model
Optimizer = Adam(learning_rate=1e-3)
Loss = CategoricalCrossentropy()
model_2.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_2 = model_2.fit(X_train_expanded, y_train_new,
                        validation_data=(X_dev_expanded, y_dev_new),
                        batch_size=128, epochs=30)

```

3. Visualization Loss & Accuracy training

```

plt.plot(history_2.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_2.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```

– Output: Hình 4a

```

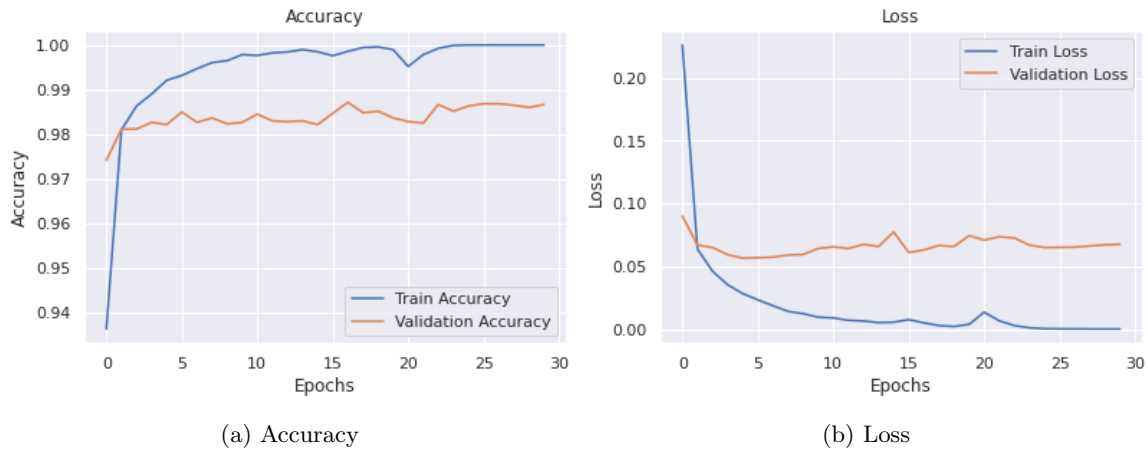
plt.plot(history_2.history['loss'], label = 'Train Loss')
plt.plot(history_2.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

– Output: Hình 4b

* Nhận xét:

- Accuracy trên tập train và tập dev đều rất cao, đặt biệt từ epoch thứ 25 accuracy tập train: = 100.0%, tập dev: ~ 98.6%. Có thể đã xảy ra hiện tượng overfitting.
- Loss trên tập train giảm nhanh và hội tụ sớm. Loss trên tập dev giảm ở những epoch đầu, sau đó liên tục biến động lên xuống và có xu hướng tăng lên. Có thể kết luận được mô hình đã bị overfitting nhẹ.
- Khi sử dụng hàm activation là *tanh* thì mô hình xảy ra hiện tượng overfitting sớm hơn mô hình sử dụng hàm activation là *relu*. Cần có các biện pháp để giảm overfitting.



Hình 4: Accuracy và Loss của model bài 2

4. Đánh giá mô hình

```

y_pred = model_2.predict(X_test_expanded)
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 = 98.8%

```

* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test cao (Accuracy test = 98.8%). Có thể thấy đây là một mô hình rất tốt.

Bài 3: Thực hiện như bài 1, nhưng thêm vào lớp Dropout với giá trị $p = 0.5$ trước lớp Dense. Huấn luyện lại mô hình và cho biết kết quả (vẽ đồ thị học của mô hình).

1. Xây dựng mạng neural bằng Keras

```

# Khởi tạo model
model_3 = Sequential()

model_3.add(Input(shape=(28, 28, 1)))
model_3.add(Conv2D(32, padding='valid', kernel_size=(3, 3),
                    activation='relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Flatten())
model_3.add(Dropout(0.5))
model_3.add(Dense(10, activation='softmax'))

model_3.summary()

```

– Output: Hình 5

* Nhận xét:

- Model có Input shape là (28,28,1)
- Model có 2 layers *Conv2D*. Layer thứ nhất có 32 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*. Layer thứ hai có 64 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*.
- Model có 2 layers *MaxPooling* có cùng kích thước là 2×2 .

- Model có 1 layer *Dropout* trước lớp Dense (lớp đầu ra) với $p = 0.5$ (tắt ngẫu nhiên 50% số node trong quá trình tính toán) để giảm hiện tượng overfitting.
- Tổng số params mà model cần phải học là 34,826 params.

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_4 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_5 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_5 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense_2 (Dense)	(None, 10)	16010

=====
 Total params: 34,826
 Trainable params: 34,826
 Non-trainable params: 0
 =====

Hình 5: Summary model

2. Huấn luyện mô hình

```

# Compile model
optimizer = Adam(learning_rate=1e-3)
Loss = CategoricalCrossentropy()
model_3.compile(optimizer=optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_3 = model_3.fit(X_train_expanded, y_train_new,
                        validation_data=(X_dev_expanded, y_dev_new),
                        batch_size=128, epochs=30)
  
```

3. Visualization Loss & Accuracy training

```

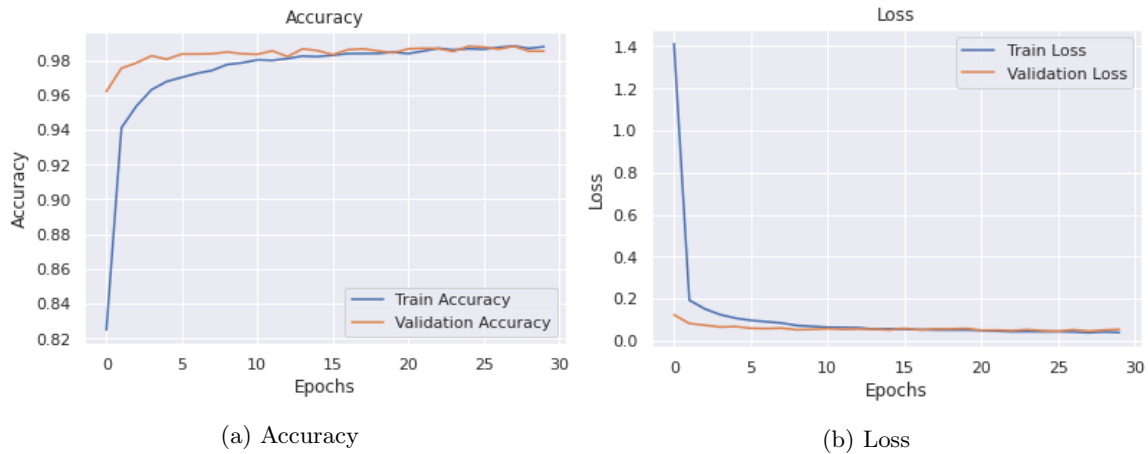
plt.plot(history_3.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_3.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
  
```

– Output: Hình 6a

```

plt.plot(history_3.history['loss'], label = 'Train Loss')
plt.plot(history_3.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
  
```

– Output: Hình 6b



Hình 6: Accuracy và Loss của model bài 3

* **Nhận xét:**

- Accuracy trên tập train và tập dev đều rất cao và xấp xỉ nhau (tập train: $\sim 98.7\%$, tập dev: $\sim 98.5\%$).
- Loss trên cả 2 tập train và dev đều giảm rất nhanh và hội tụ rất sớm (khoảng epoch thứ 5).
- Nhìn vào kết quả của đồ thị học có thể thấy mô hình đã không còn bị overfitting.
- Bằng cách thêm vào 1 lớp *Dropout* cho mô hình thì mô hình bài 3 đã khắc phục được hiện tượng overfitting của mô hình ở bài 1.

4. Đánh giá mô hình

```

y_pred = model_3.predict(X_test_expanded)
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 = 98.97%

```

* **Nhận xét:**

- Accuracy của mô hình khi kiểm tra trên tập test cao (Accuracy test = 98.97%). Có thể thấy đây là một mô hình rất tốt. Tốt hơn 2 mô hình ở bài 1 và bài 2 vì đã khắc phục được hiện tượng overfitting.

Bài 4: Hiện thực lại mô hình LeNET-5 theo hướng dẫn ở mục 4. Cho biết kết quả độ chính xác.

1. Xây dựng mạng neural bằng Keras

```

# Khởi tạo model
model_4 = Sequential()

model_4.add(Conv2D(filters=6, kernel_size=(5,5), padding='same',
                    activation='relu', input_shape=(28,28,1)))
model_4.add(AveragePooling2D(strides=2, pool_size=(2, 2)))

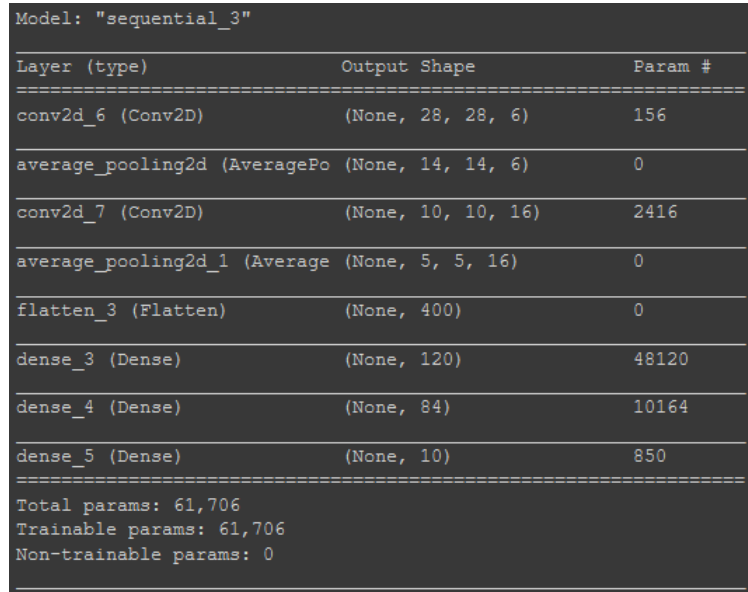
model_4.add(Conv2D(filters=16, kernel_size=(5,5), padding='valid',
                    activation='relu'))
model_4.add(AveragePooling2D(strides=2, pool_size=(2, 2)))

model_4.add(Flatten())
model_4.add(Dense(120, activation='relu'))
model_4.add(Dense(84, activation='relu'))
model_4.add(Dense(10, activation='softmax'))

model_4.summary()

```

– Output: Hình 9



```
Model: "sequential_3"
Layer (type)                 Output Shape                 Param #
-----
conv2d_6 (Conv2D)            (None, 28, 28, 6)           156
average_pooling2d (AveragePo (None, 14, 14, 6)           0
conv2d_7 (Conv2D)            (None, 10, 10, 16)          2416
average_pooling2d_1 (Average (None, 5, 5, 16)           0
flatten_3 (Flatten)          (None, 400)                  0
dense_3 (Dense)              (None, 120)                  48120
dense_4 (Dense)              (None, 84)                   10164
dense_5 (Dense)              (None, 10)                   850
-----
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

Hình 7: Summary model

* **Nhận xét:**

- Model có Input shape là (28,28,1)
- Model có 2 layers *Conv2D*. Layer thứ nhất có 6 filter với mỗi filter có kích thước 5×5 và sử dụng hàm activation là *relu*. Layer thứ hai có 16 filter với mỗi filter có kích thước 5×5 và sử dụng hàm activation là *relu*.
- Model có 2 layers *AveragePooling* có cùng kích thước là 2×2 và có *strides* (bước trượt) là 2.
- Tổng số params mà model cần phải học là 61,706 params.

2. Huấn luyện mô hình

```
# Compile model
Optimizer = Adam(learning_rate=1e-3)
Loss = CategoricalCrossentropy()
model_4.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_4 = model_4.fit(X_train_expanded, y_train_new,
                        validation_data=(X_dev_expanded, y_dev_new),
                        batch_size=128, epochs=30)
```

3. Visualization Loss & Accuracy training

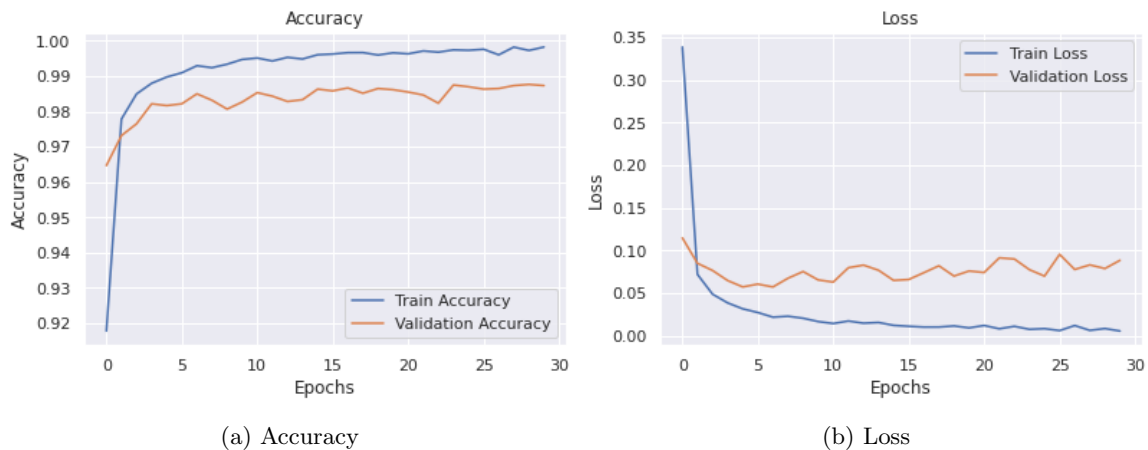
```
plt.plot(history_4.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_4.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 8a

```
plt.plot(history_4.history['loss'], label = 'Train Loss')
plt.plot(history_4.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()
```

– Output: Hình 8b



Hình 8: Accuracy và Loss của model bài 4

* **Nhận xét:**

- Accuracy trên tập train và tập dev đều rất cao (tập train: $\sim 99.8\%$, tập dev: $\sim 98.7\%$).
- Loss trên tập train giảm nhanh và hội tụ sớm. Loss trên tập dev giảm ở những epoch đầu, sau đó liên tục biến động lên xuống và có xu hướng tăng lên. Có thể kết luận được mô hình đã bị overfitting nhẹ.

4. Đánh giá mô hình

```
y_pred = model_4.predict(X_test_expanded)
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 = 98.93%
```

* **Nhận xét:**

- Accuracy của mô hình khi kiểm tra trên tập test cao (Accuracy test = 98.93%). Có thể thấy đây là một mô hình rất tốt. Mặc dù, mô hình bị overfitting nhẹ nhưng nhìn chung thì mô hình vẫn rất tốt.

Bài 5: Lưu lại mô hình ở bài 4 thành file h5

```
# Save model LeNET-5
model_4.save('LeNET-5.h5')
```

Bài 6:* Hiện thực mạng AlexNET với bộ dữ liệu MNIST.

1. Xây dựng mạng neural bằng Keras

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

model_5.add(Conv2D(filters=96, kernel_size=(12,12), padding='same',
                    activation='relu', input_shape=(28,28,1), strides=(4,4)))
model_5.add(MaxPooling2D(strides=(2,2), pool_size=(3, 3), padding='same'))

model_5.add(Conv2D(filters=256, kernel_size=(5,5), padding='same',
                    activation='relu', strides=(1,1)))
```

```

model_5.add(MaxPooling2D(strides=(2,2), pool_size=(3, 3), padding='same'))

model_5.add(Conv2D(filters=384, kernel_size=(3,3), activation='relu',
                  strides=(1,1), padding='same'))

model_5.add(Conv2D(filters=384, kernel_size=(3,3), activation='relu',
                  strides=(1,1), padding='same'))

model_5.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu',
                  strides=(1,1), padding='same'))
model_5.add(MaxPooling2D(strides=(2,2), pool_size=(3, 3), padding='same'))
model_5.add(Flatten())
model_5.add(Dropout(0.5))
model_5.add(Dense(4096, activation='relu'))
model_5.add(Dense(4096, activation='relu'))
model_5.add(Dense(10, activation='softmax'))

model_5.summary()

```

– Output: Hình 9

Model: "sequential_4"		
Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 7, 7, 96)	13920
max_pooling2d_6 (MaxPooling2D)	(None, 4, 4, 96)	0
conv2d_9 (Conv2D)	(None, 4, 4, 256)	614656
max_pooling2d_7 (MaxPooling2D)	(None, 2, 2, 256)	0
conv2d_10 (Conv2D)	(None, 2, 2, 384)	885120
conv2d_11 (Conv2D)	(None, 2, 2, 384)	1327488
conv2d_12 (Conv2D)	(None, 2, 2, 256)	884992
max_pooling2d_8 (MaxPooling2D)	(None, 1, 1, 256)	0
flatten_4 (Flatten)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 4096)	1052672
dense_7 (Dense)	(None, 4096)	16781312
dense_8 (Dense)	(None, 10)	40970
Total params: 21,601,130		
Trainable params: 21,601,130		
Non-trainable params: 0		

Hình 9: Summary model

* Nhận xét:

- Model có Input shape là (28,28,1)
- Model có 5 layers *Conv2D*. Layer thứ nhất có 96 filter với mỗi filter có kích thước 12×12 , *strides* là 4, padding là same và sử dụng hàm activation là *relu*. Layer thứ 2 có 256 filter với mỗi filter có kích thước 5×5 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*. Layer thứ 3 có 384 filter với mỗi filter có kích thước 3×3 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*. Layer thứ 4 có 384 filter với mỗi filter có kích thước 3×3 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*. Layer thứ 5 có 256 filter với mỗi filter có kích thước 3×3 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*.

- Model có 3 layers *MaxPooling* có cùng kích thước là 2×2 , có *strides* (bước trượt) là 2 và có padding là same.
- Model có 1 layer *Dropout* sau lớp Flatten (lớp duỗi thành vector 1 chiều) với $p = 0.5$ (tắt ngẫu nhiên 50% số node trong quá trình tính toán) để giảm hiện tượng overfitting.
- Tổng số params mà model cần phải học là 21,601,130 params.

2. Huấn luyện mô hình

```
# Compile model
Optimizer = Adam(learning_rate=1e-4)
Loss = CategoricalCrossentropy()
model_5.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_5 = model_5.fit(X_train_expanded, y_train_new,
                        validation_data=(X_dev_expanded, y_dev_new),
                        batch_size=128, epochs=30)
```

3. Visualization Loss & Accuracy training

```
plt.plot(history_5.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_5.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 10a

```
plt.plot(history_5.history['loss'], label = 'Train Loss')
plt.plot(history_5.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 10b



Hình 10: Accuracy và Loss của model bài 6

* Nhận xét:

- Accuracy trên tập train và tập dev đều rất cao (tập train: $\sim 99.8\%$, tập dev: $\sim 97.9\%$).

- Loss trên tập train giảm nhanh và hội tụ sớm. Loss trên tập dev giảm ở những epoch đầu, sau đó liên tục biến động lên xuống và có xu hướng tăng lên. Có thể kết luận được mô hình đã bị overfitting nhẹ.

4. Đánh giá mô hình

```
y_pred = model_5.predict(X_test_expanded)
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 = 98.74%
```

* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test cao (Accuracy test = 98.74%). Có thể thấy đây là một mô hình rất tốt. Mặc dù, mô hình bị overfitting nhẹ nhưng nhìn chung thì mô hình vẫn rất tốt.

Bài 7:* Thực hiện các yêu cầu trên đối với bộ dữ liệu CIFAR10.

Set up

1. Import các thư viện cần thiết

- Các thư viện cần thiết sử dụng lại từ phần [Set up](#).

```
from keras.regularizers import l2
from keras.layers import BatchNormalization
from keras.datasets.cifar10 import load_data
```

2. Load bộ dữ liệu MNIST

```
(X_train_extra, y_train_extra), (X_test_extra, y_test_extra) = load_data()
```

3. Chuẩn bị dữ liệu

```
X_train_extra, X_dev_extra, y_train_extra, y_dev_extra = train_test_split(X_train_extra,
                                                                            y_train_extra,
                                                                            test_size=0.1)
```

```
print('X_train_extra = {}'.format(X_train_extra.shape))
# Output: X_train_extra = (45000, 32, 32, 3)
print('y_train_extra = {}'.format(y_train_extra.shape))
# Output: y_train_extra = (45000, 1)
print('X_dev_extra = {}'.format(X_dev_extra.shape))
# Output: X_dev_extra = (5000, 32, 32, 3)
print('y_dev_extra = {}'.format(y_dev_extra.shape))
# Output: y_dev_extra = (5000, 1)
print('X_test_extra = {}'.format(X_test_extra.shape))
# Output: X_test_extra = (10000, 32, 32, 3)
print('y_test_extra = {}'.format(y_test_extra.shape))
# Output: y_test_extra = (10000, 1)
```

- X_train_extra gồm có 45,000 bức ảnh màu (số channel = 3) với kích thước mỗi bức ảnh màu là 32×32 pixel dùng để train model, y_train_extra là các nhãn tương ứng X_train_extra.
- X_dev_extra gồm có 5,000 bức ảnh màu (số channel = 3) với kích thước mỗi bức ảnh màu là 32×32 pixel dùng để test model, y_dev_extra là các nhãn tương ứng của X_dev_extra.
- X_test_extra gồm có 10,000 bức ảnh màu (số channel = 3) với kích thước mỗi bức ảnh màu là 32×32 pixel dùng để test model, y_test_extra là các nhãn tương ứng của X_test_extra.

```
y_train_extra_new = to_categorical(y_train_extra, num_classes = 10)
y_dev_extra_new = to_categorical(y_dev_extra, num_classes = 10)
```

```

print('X_train_extra = {}'.format(X_train_extra.shape))
# Output: X_train_extra = (45000, 32, 32, 3)
print('y_train_extra_new = {}'.format(y_train_extra_new.shape))
# Output: y_train_extra_new = (45000, 10)
print('X_dev_extra = {}'.format(X_dev_extra.shape))
# Output: X_dev_extra = (5000, 32, 32, 3)
print('y_dev_extra_new = {}'.format(y_dev_extra_new.shape))
# Output: y_dev_extra_new = (5000, 10)
print('X_test_extra = {}'.format(X_test_extra.shape))
# Output: X_test_extra = (10000, 32, 32, 3)
print('y_test_extra = {}'.format(y_test_extra.shape))
# Output: y_test_extra = (10000, 1)

```

Bài 7.1: Hiện thực mô hình ở phần 3, sử dụng hàm activation relu cho 2 lớp CONV. Thực hiện huấn luyện lại mô hình và xem kết quả? Cho biết độ chính xác và đồ thị học của mô hình.

1. Xây dựng mạng neural bằng Keras

```

# Khởi tạo model
model_6_1 = Sequential()

model_6_1.add(Input(shape=(32, 32, 3)))
model_6_1.add(Conv2D(64, padding='same', kernel_size=(3, 3),
                    activation='relu',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_1.add(BatchNormalization())
model_6_1.add(MaxPooling2D(pool_size=(2, 2)))

model_6_1.add(Conv2D(128, padding='same', kernel_size=(3, 3),
                    activation='relu',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_1.add(BatchNormalization())
model_6_1.add(MaxPooling2D(pool_size=(2, 2)))

model_6_1.add(Conv2D(256, padding='same', kernel_size=(3, 3),
                    activation='relu',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_1.add(Dropout(0.5))
model_6_1.add(BatchNormalization())
model_6_1.add(MaxPooling2D(pool_size=(2, 2)))

model_6_1.add(Flatten())
model_6_1.add(Dropout(0.5))
model_6_1.add(Dense(1024, activation='relu'))
model_6_1.add(Dropout(0.5))
model_6_1.add(Dense(512, activation='relu'))
model_6_1.add(Dropout(0.5))
model_6_1.add(Dense(10, activation='softmax'))

model_6_1.summary()

```

– Output: Hình 11

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (Batch Normalization)	(None, 32, 32, 64)	256
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_14 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_1 (Batch Normalization)	(None, 16, 16, 128)	512
max_pooling2d_10 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_15 (Conv2D)	(None, 8, 8, 256)	295168
dropout_2 (Dropout)	(None, 8, 8, 256)	0
batch_normalization_2 (Batch Normalization)	(None, 8, 8, 256)	1024
max_pooling2d_11 (MaxPooling2D)	(None, 4, 4, 256)	0
flatten_5 (Flatten)	(None, 4096)	0
dropout_3 (Dropout)	(None, 4096)	0
dense_9 (Dense)	(None, 1024)	4195328
dropout_4 (Dropout)	(None, 1024)	0
dense_10 (Dense)	(None, 512)	524800
dropout_5 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 10)	5130
Total params: 5,097,866		
Trainable params: 5,096,970		
Non-trainable params: 896		

Hình 11: Summary model

* **Nhận xét:**

- Model có Input shape là (32,32,3)
- Model có 3 layers *Conv2D*. Layer thứ nhất có 64 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*. Layer thứ 2 có 128 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*. Layer thứ 3 có 256 filter với mỗi filter có kích thước 3×3 và sử dụng hàm activation là *relu*.
- Model có 3 layers *MaxPooling* có cùng kích thước là 2×2 .
- Model có sử dụng kĩ thuật Regularization với dạng chuẩn hóa l2 trong các layer *Conv2D* để giảm overfitting.
- Model có 3 layers *Batch Normalization* để chuẩn hóa dữ liệu, ổn định quá trình học của mô hình.
- Model có 4 layer *Dropout* với $p = 0.5$ (tắt ngẫu nhiên 50% số node trong quá trình tính toán) để giảm hiện tượng overfitting.
- Tổng số params mà model cần phải học là 5,096,970 params.

2. Huấn luyện mô hình

```
# Compile model
Optimizer = Adam(learning_rate=1e-4)
Loss = CategoricalCrossentropy()
model_6_1.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_6_1 = model_6_1.fit(X_train_extra, y_train_extra_new,
```

```
validation_data=(X_dev_extra, y_dev_extra_new),
batch_size=128, epochs=50)
```

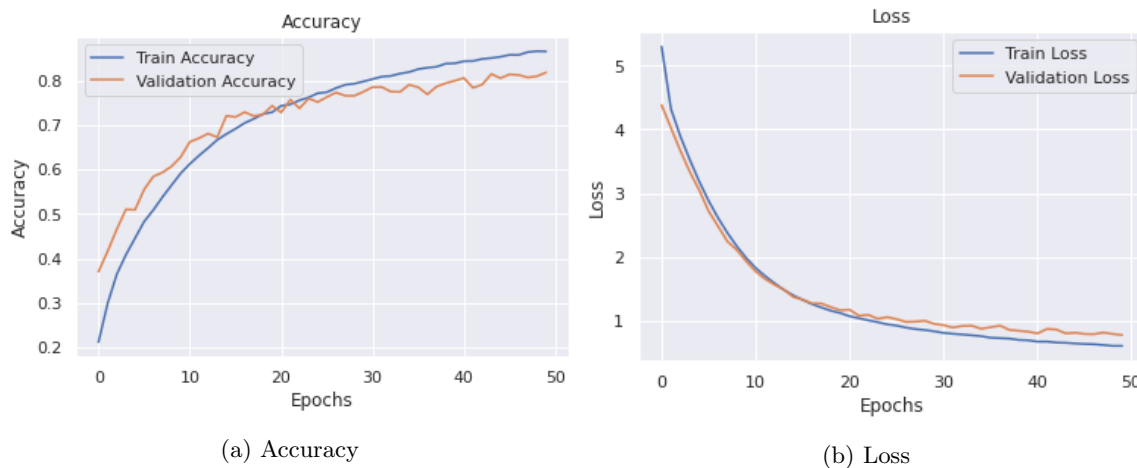
3. Visualization Loss & Accuracy training

```
plt.plot(history_6_1.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_6_1.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 12a

```
plt.plot(history_6_1.history['loss'], label = 'Train Loss')
plt.plot(history_6_1.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 12b



Hình 12: Accuracy và Loss của model bài 7.1

* Nhận xét:

- Accuracy trên tập train và tập dev đều khá cao (tập train: $\sim 86.4\%$, tập dev: $\sim 80.1\%$).
- Loss trên cả 2 tập train và dev giảm đều và ổn định qua các epoch.

4. Đánh giá mô hình

```
y_pred = model_6_1.predict(X_test_extra)
y_pred = np.argmax(y_pred, axis = -1)
accuracy = round(accuracy_score(y_test_extra, y_pred)*100,2)
print('Accuracy test = {}'.format(accuracy))
# Output: Accuracy test = 82.37%
```

* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test khá cao (Accuracy test = 82.37%). Mô hình này khá tốt.

Bài 7.2: Hiện thực lại mô hình LeNET-5 theo hướng dẫn ở mục 4. Cho biết kết quả độ chính xác.

1. Xây dựng mạng neural bằng Keras

```
# Khởi tạo model
model_6_2 = Sequential()

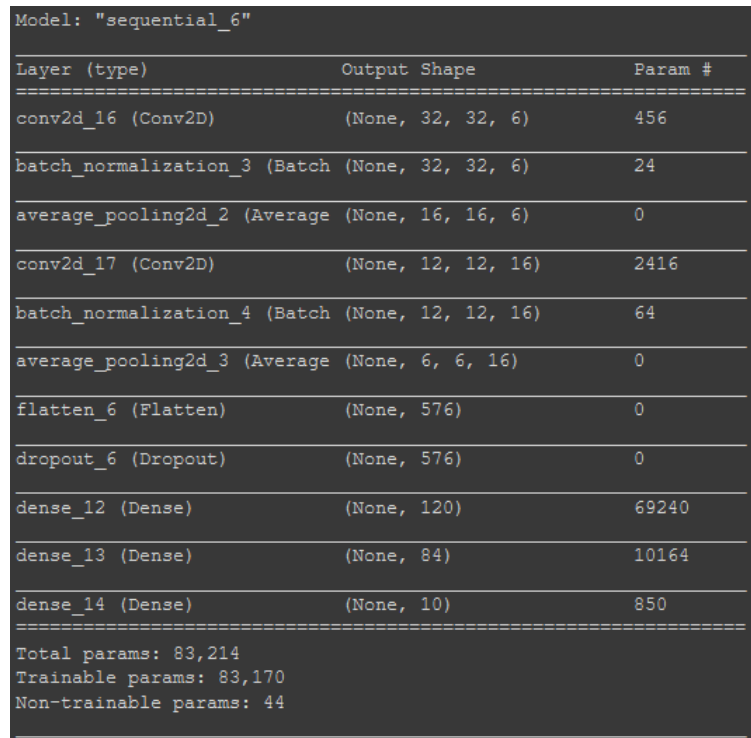
model_6_2.add(Conv2D(filters=6, kernel_size=(5,5), padding='same',
                    activation='relu', input_shape=(32,32,3),
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_2.add(BatchNormalization())
model_6_2.add(AveragePooling2D(strides=2, pool_size=(2, 2)))

model_6_2.add(Conv2D(filters=16, kernel_size=(5,5), padding='valid',
                    activation='relu',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_2.add(BatchNormalization())
model_6_2.add(AveragePooling2D(strides=2, pool_size=(2, 2)))

model_6_2.add(Flatten())
model_6_2.add(Dropout(0.5))
model_6_2.add(Dense(120, activation='relu'))
model_6_2.add(Dense(84, activation='relu'))
model_6_2.add(Dense(10, activation='softmax'))

model_6_2.summary()
```

– Output: Hình 13



Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 32, 32, 6)	456
batch_normalization_3 (Batch Normalization)	(None, 32, 32, 6)	24
average_pooling2d_2 (Average Pooling)	(None, 16, 16, 6)	0
conv2d_17 (Conv2D)	(None, 12, 12, 16)	2416
batch_normalization_4 (Batch Normalization)	(None, 12, 12, 16)	64
average_pooling2d_3 (Average Pooling)	(None, 6, 6, 16)	0
flatten_6 (Flatten)	(None, 576)	0
dropout_6 (Dropout)	(None, 576)	0
dense_12 (Dense)	(None, 120)	69240
dense_13 (Dense)	(None, 84)	10164
dense_14 (Dense)	(None, 10)	850
Total params: 83,214		
Trainable params: 83,170		
Non-trainable params: 44		

Hình 13: Summary model

* **Nhận xét:**

- Model có Input shape là (32,32,3)
- Model có 2 layers *Conv2D*. Layer thứ nhất có 6 filter với mỗi filter có kích thước 5×5 và sử dụng hàm activation là *relu*. Layer thứ hai có 16 filter với mỗi filter có kích thước 5×5 và sử dụng hàm activation là *relu*.

- Model có 2 layers *AveragePooling* có cùng kích thước là 2×2 và có *strides* (bước trượt) là 2.
- Model có sử dụng kỹ thuật Regularization với dạng chuẩn hóa l2 trong các layer *Conv2D* để giảm overfitting.
- Model có 2 layers *Batch Normalization* để chuẩn hóa dữ liệu, ổn định quá trình học của mô hình.
- Model có 1 layer *Dropout* với $p = 0.5$ (tắt ngẫu nhiên 50% số node trong quá trình tính toán) để giảm hiện tượng overfitting.
- Tổng số params mà model cần phải học là 83,170 params.

2. Huấn luyện mô hình

```
# Compile model
Optimizer = Adam(learning_rate=1e-4)
Loss = CategoricalCrossentropy()
model_6_1.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_6_1 = model_6_1.fit(X_train_extra, y_train_extra_new,
                             validation_data=(X_dev_extra, y_dev_extra_new),
                             batch_size=128, epochs=50)
```

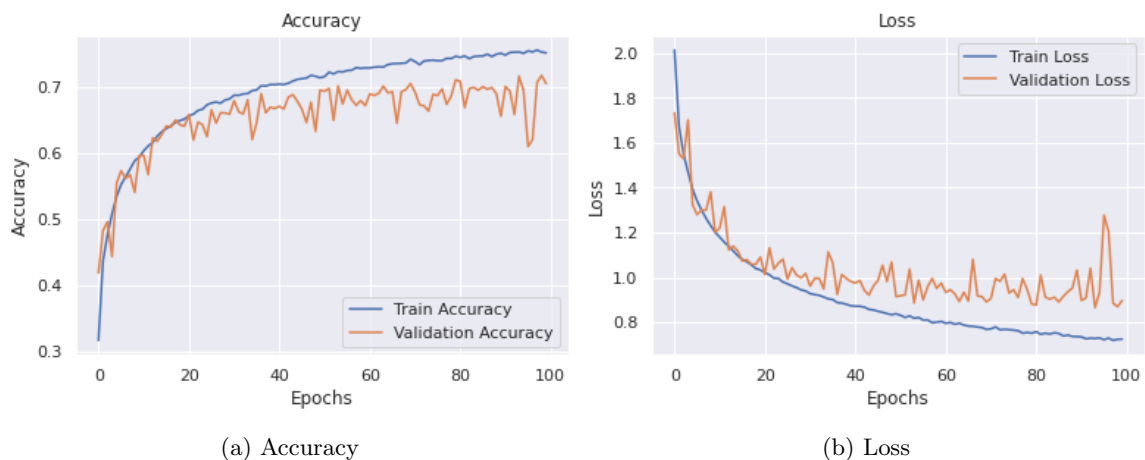
3. Visualization Loss & Accuracy training

```
plt.plot(history_6_2.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_6_2.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 14a

```
plt.plot(history_6_2.history['loss'], label = 'Train Loss')
plt.plot(history_6_2.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 14b



Hình 14: Accuracy và Loss của model bài 7.2

* Nhận xét:

- Accuracy trên tập train và tập dev đều khá thấp (tập train: $\sim 75.2\%$, tập dev: $\sim 70.5\%$). Accuracy trên tập dev dao động lên xuống mạnh qua các epoch (lúc tăng, lúc giảm) nhưng cuối cùng nó vẫn tăng.
- Loss trên tập train giảm đều và ổn định qua các epoch. Loss trên tập dev dao động lên xuống mạnh qua các epoch (lúc tăng, lúc giảm) nhưng cuối cùng nó vẫn giảm.

4. Đánh giá mô hình

```
y_pred = model_6_2.predict(X_test_extra)
y_pred = np.argmax(y_pred, axis = -1)
accuracy = round(accuracy_score(y_test_extra, y_pred)*100,2)
print('Accuracy test = {}'.format(accuracy))
# Output: Accuracy test = 70.78%
```

* Nhận xét:

- Accuracy của mô hình khi kiểm tra trên tập test khá thấp (Accuracy test = 70.78%). Kết quả của mô hình tạm chấp nhận được.

Bài 7.3: Hiện thực mạng AlexNET với bộ dữ liệu CIFAR-10.

1. Xây dựng mạng neural bằng Keras

```
# Khởi tạo model
model_6_3 = Sequential()

model_6_3.add(Conv2D(filters=96, kernel_size=(11,11), padding='same',
                    activation='relu', input_shape=(32,32,3), strides=(4,4),
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_3.add(BatchNormalization())
model_6_3.add(MaxPooling2D(strides=(2,2), pool_size=(3, 3), padding='same'))

model_6_3.add(Conv2D(filters=256, kernel_size=(5,5), padding='same',
                    activation='relu', strides=(1,1),
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_3.add(BatchNormalization())
model_6_3.add(Dropout(0.5))
model_6_3.add(MaxPooling2D(strides=(2,2), pool_size=(3, 3), padding='same'))

model_6_3.add(Conv2D(filters=384, kernel_size=(3,3), activation='relu',
                    strides=(1,1), padding='same',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_3.add(BatchNormalization())
model_6_3.add(Dropout(0.5))

model_6_3.add(Conv2D(filters=384, kernel_size=(3,3), activation='relu',
                    strides=(1,1), padding='same',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_3.add(BatchNormalization())
model_6_3.add(Dropout(0.5))

model_6_3.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu',
                    strides=(1,1), padding='same',
                    kernel_regularizer = l2 (0.01),
                    bias_regularizer = l2 (0.01)))
model_6_3.add(BatchNormalization())
model_6_3.add(Dropout(0.5))
model_6_3.add(MaxPooling2D(strides=(2,2), pool_size=(3, 3), padding='same'))

model_6_3.add(Flatten())
```

```

model_6_3.add(Dropout(0.5))
model_6_3.add(Dense(4096, activation='relu'))
model_6_3.add(Dropout(0.5))
model_6_3.add(Dense(4096, activation='relu'))
model_6_3.add(Dropout(0.5))
model_6_3.add(Dense(10, activation='softmax'))

model_6_3.summary()

```

– Output: Hình 15

```

Model: "sequential_7"

```

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 8, 8, 96)	34944
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 96)	384
max_pooling2d_12 (MaxPooling2D)	(None, 4, 4, 96)	0
conv2d_19 (Conv2D)	(None, 4, 4, 256)	614656
batch_normalization_6 (Batch Normalization)	(None, 4, 4, 256)	1024
dropout_7 (Dropout)	(None, 4, 4, 256)	0
max_pooling2d_13 (MaxPooling2D)	(None, 2, 2, 256)	0
conv2d_20 (Conv2D)	(None, 2, 2, 384)	885120
batch_normalization_7 (Batch Normalization)	(None, 2, 2, 384)	1536
dropout_8 (Dropout)	(None, 2, 2, 384)	0
conv2d_21 (Conv2D)	(None, 2, 2, 384)	1327488
batch_normalization_8 (Batch Normalization)	(None, 2, 2, 384)	1536
dropout_9 (Dropout)	(None, 2, 2, 384)	0
conv2d_22 (Conv2D)	(None, 2, 2, 256)	884992
batch_normalization_9 (Batch Normalization)	(None, 2, 2, 256)	1024
dropout_10 (Dropout)	(None, 2, 2, 256)	0
max_pooling2d_14 (MaxPooling2D)	(None, 1, 1, 256)	0
flatten_7 (Flatten)	(None, 256)	0
dropout_11 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 4096)	1052672
dropout_12 (Dropout)	(None, 4096)	0
dense_16 (Dense)	(None, 4096)	16781312
dropout_13 (Dropout)	(None, 4096)	0
dense_17 (Dense)	(None, 10)	40970
Total params: 21,627,658		
Trainable params: 21,624,906		
Non-trainable params: 2,752		

Hình 15: Summary model

* Nhận xét:

- Model có Input shape là (32,32,3)
- Model có 5 layers *Conv2D*. Layer thứ nhất có 96 filter với mỗi filter có kích thước 12×12 , *strides* là 4, padding là same và sử dụng hàm activation là *relu*. Layer thứ 2 có 256 filter với mỗi filter có kích thước 5×5 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*. Layer thứ 3 có 384 filter với mỗi filter có kích thước

3×3 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*. Layer thứ 4 có 384 filter với mỗi filter có kích thước 3×3 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*. Layer thứ 5 có 256 filter với mỗi filter có kích thước 3×3 , *strides* là 1, padding là same và sử dụng hàm activation là *relu*.

- Model có 3 layers *MaxPooling* có cùng kích thước là 2×2 , có *strides* (bước trượt) là 2 và có padding là same.
- Model có sử dụng kĩ thuật Regularization với dạng chuẩn hóa l2 trong các layer *Conv2D* để giảm overfitting.
- Model có 5 layers *Batch Normalization* để chuẩn hóa dữ liệu, ổn định quá trình học của mô hình.
- Model có 7 layer *Dropout* với $p = 0.5$ (tắt ngẫu nhiên 50% số node trong quá trình tính toán) để giảm hiện tượng overfitting.
- Tổng số params mà model cần phải học là 21,624,906 params.

2. Huấn luyện mô hình

```
# Compile model
Optimizer = Adam(learning_rate=1e-4)
Loss = CategoricalCrossentropy()
model_6_3.compile(optimizer=Optimizer, loss=Loss, metrics=['accuracy'])

# Training model
history_6_3 = model_6_3.fit(X_train_extra, y_train_extra_new,
                             validation_data=(X_dev_extra, y_dev_extra_new),
                             batch_size=64, epochs=30)
```

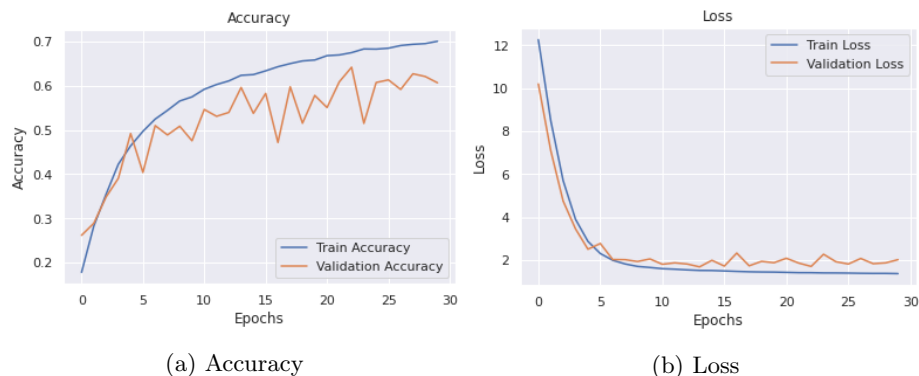
3. Visualization Loss & Accuracy training

```
plt.plot(history_6_3.history['accuracy'], label = 'Train Accuracy')
plt.plot(history_6_3.history['val_accuracy'], label = 'Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

– Output: Hình 16a

```
plt.plot(history_6_3.history['loss'], label = 'Train Loss')
plt.plot(history_6_3.history['val_loss'], label = 'Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

– Output: Hình 16b



Hình 16: Accuracy và Loss của model bài 7.3

* **Nhận xét:**

- Accuracy trên tập train và tập dev đều khá thấp (tập train: $\sim 70.1\%$, tập dev: $\sim 60.7\%$). Accuracy trên tập dev dao động lên xuống mạnh qua các epoch (lúc tăng, lúc giảm) nhưng cuối cùng nó vẫn tăng.
- Loss trên tập train giảm đều và ổn định qua các epoch. Loss trên tập dev dao động lên xuống nhẹ qua các epoch (lúc tăng, lúc giảm) nhưng cuối cùng nó vẫn giảm. Loss của mô hình vẫn còn cao (~ 2).

4. Đánh giá mô hình

```
y_pred = model_6_3.predict(X_test_extra)
y_pred = np.argmax(y_pred, axis = -1)
accuracy = round(accuracy_score(y_test_extra, y_pred)*100,2)
print('Accuracy test = {}'.format(accuracy))
# Output: Accuracy test = 60.76%
```

* **Nhận xét:**

- Accuracy của mô hình khi kiểm tra trên tập test khá thấp (Accuracy test = 60.76%). Mô hình này không được tốt lắm.