Họ và tên: **Nguyễn Văn Mạnh**

Mã số sinh viên: **102200024**

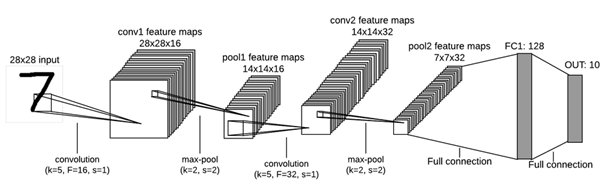
Lớp: **20T1**

**Đề số 1**

**NỘI DUNG NỘP BÀI**

# Câu 1 – CNN

## Câu 1 a. Đề xuất mô hình và show feature map



Code của mô hình

import torch

import torch.nn as nn

import torch.nn.functional as F

import matplotlib.pyplot as plt

class SimpleCNN(nn.Module):

    def \_\_init\_\_(self):

        super(SimpleCNN, self).\_\_init\_\_()

        # KHAI BÁO CÁO LỚP

        # conv1 có in\_channels = 1 (hiện tại là ảnh gray , có thể chỉnh lên 3 được )

        # out\_channels là F (như trong hình)

        # kernel\_size là k (như trong hình) (kernel\_size=3 là kích thướt 3x3 , là mấy kernel phía trên thầy cho,

        # còn dưới kiến trúc kernel\_size k = 5 thì không dùng, đổi lại thành 3 để dùng mấy cái ma trận phía trên)

        # stride là s (như trong hình)

        # padding không có nhưng mà ta thấy giữ nguyên được W và H sau khi qua Conv thì p = s hoặc p = same

        self.conv1 = nn.Conv2d(in\_channels=1, out\_channels=16, kernel\_size=3, stride=1, padding=1) # padding = same ?

        self.conv2 = nn.Conv2d(in\_channels=16, out\_channels=32, kernel\_size=3, stride=1, padding=1)

        self.fc1 = nn.Linear(32 \* 7 \* 7, 128)

        self.fc2 = nn.Linear(128, 10) # 10 nhãn

        self.pool = nn.MaxPool2d(kernel\_size=2, stride=2, padding=0)

        self.feature\_maps = []

        self.flatten = nn.Flatten()

    # SỬ DỤNG CÁC LỚP

    def forward(self, x):

        x = F.relu(self.conv1(x))

        self.feature\_maps.append(x)

        x = self.pool(x)

        x = F.relu(self.conv2(x))

        self.feature\_maps.append(x)

        x = self.pool(x)

        x = x.view(-1, 32 \* 7 \* 7) # flatten

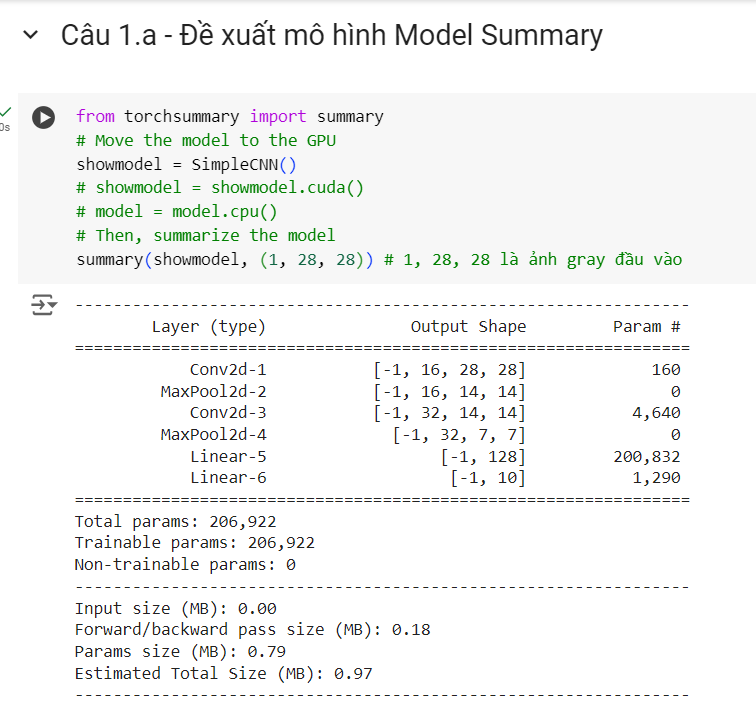
        x = F.relu(self.fc1(x))

        x = self.fc2(x)

        return x

# Khởi tạo mô hình

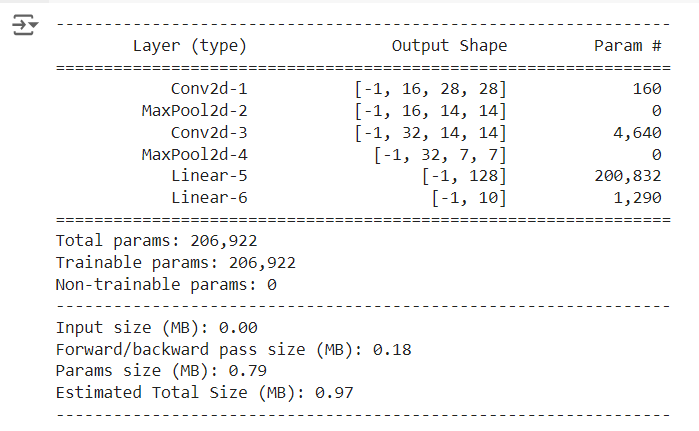
model = SimpleCNN()



#### Tính toán

* Input tensor : 28\*28\*1
* Cho Input X (W, H, D) = (28,28,1) qua Conv1(F=16,k\*k=3\*3,D=1,s=1,p=1)
* Cho Input X (W, H, D) = qua Max-Pooling (k=2,s=2,p=0)
* Cho Input X (W, H, D) = qua Conv2(F=32,k\*k=3\*3,D=16,s=1,p=1)
* Cho Input X (W, H, D) = qua Max-Pooling2 (k=2,s=2,p=0)
* Cho Input X (W, H, D) = qua Flatten thu được Vector (7\*7\*32,1) = **(1568,1)**
* Cho Input Vector X(1568,1) qua Fully connected1 (128) thu được Vector **(128,1)**
* Cho Input Vector X(,1) qua Fully connected2 (10) thu được Vector **(10,1)**
* Tổng số parameters = = **206922**

#### Kiến trúc thu gọn



### Show feature map

import matplotlib.pyplot as plt

def plot\_feature\_map(feature\_maps, layer\_index=0, feature\_map\_index=8):

    # Lấy feature map của layer đầu tiên

    fmap = feature\_maps[layer\_index].detach().cpu().numpy()

    # Kiểm tra số lượng filters để đảm bảo feature map thứ 8 tồn tại

    if feature\_map\_index >= fmap.shape[1]:

        print(f"Layer {layer\_index+1} chỉ có {fmap.shape[1]} filters, không thể vẽ feature map thứ {feature\_map\_index+1}")

        return

    # Lấy feature map thứ 8

    fmap\_to\_plot = fmap[0, feature\_map\_index]

    # Vẽ ảnh của feature map thứ 8

    plt.figure(figsize=(4, 4))

    plt.imshow(fmap\_to\_plot)

    # plt.imshow(fmap\_to\_plot, cmap='gray')

    plt.title(f'Layer {layer\_index+1} Feature Map {feature\_map\_index} - Tháng 8 ')

    plt.axis('off')

    plt.show()

# Khởi tạo mô hình và tải trọng số đã huấn luyện (nếu có)

# model = SimpleCNN()

# Đọc ảnh từ file

image\_path = './cat.jpg'  # Thay thế bằng đường dẫn tới ảnh của bạn

image = load\_image(image\_path)

# Xóa feature maps cũ trước khi forward

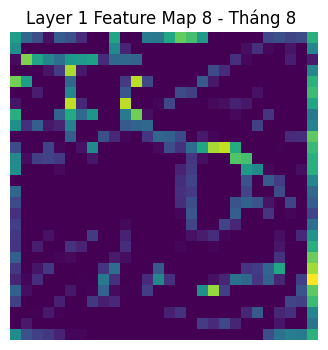
model.feature\_maps = []

# Forward qua mô hình

output = model(image.unsqueeze(0))

# Hiển thị feature map thứ 8 của layer đầu tiên

plot\_feature\_map(model.feature\_maps, layer\_index=0, feature\_map\_index=8)



## Câu 1.b. Định nghĩa hàm kích hoạt GELU và hàm optimizer : Tanh

import numpy as np

class TanhOptimizer:

    def \_\_init\_\_(self, params, lr=0.01):

        self.params = list(params)

        self.lr = lr

    def step(self):

        for param in self.params:

            if param.grad is not None:

                # Cập nhật giá trị tham số dựa trên gradient và learning rate

                param.data -= self.lr \* param.grad

    def zero\_grad(self):

        for param in self.params:

            if param.grad is not None:

                param.grad = 0.0

class Parameter:

    def \_\_init\_\_(self, data):

        self.data = data

        self.grad = None

    def compute\_grad(self):

        # Tính gradient của hàm tanh

        self.grad = 1 - np.tanh(self.data)2

## Câu 1.c Hiện thị vector flatten sau khi làm phẳng

# Tải và xử lý ảnh

image\_path = './cat.jpg'  # Thay thế bằng đường dẫn tới ảnh của bạn

image = load\_image(image\_path)

# Forward pass để lấy output của lớp flatten

with torch.no\_grad():

    x = model.pool(torch.relu(model.conv1(image)))

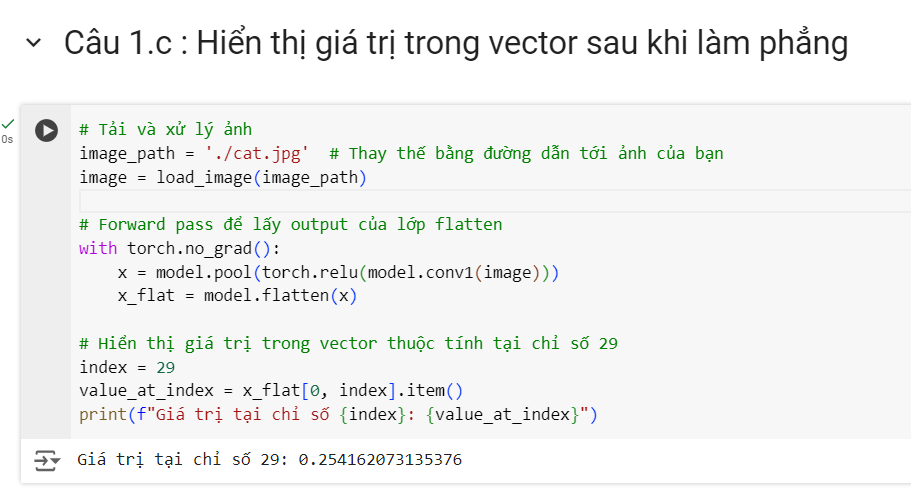
    x\_flat = model.flatten(x)

# Hiển thị giá trị trong vector thuộc tính tại chỉ số 29

index = 29

value\_at\_index = x\_flat[0, index].item()

print(f"Giá trị tại chỉ số {index}: {value\_at\_index}")



## Câu 1.d. Chọn hàm mất mát

def custom\_mse\_loss(output, target):

    loss = torch.mean((output - target) 2)

    return loss

**Lý do chọn hàm mất mát Mean Squared Error (MSE)**

1. Đơn giản và dễ hiểu:

2. Khả năng phạt các lỗi lớn:

3. Liên tục và khả vi:

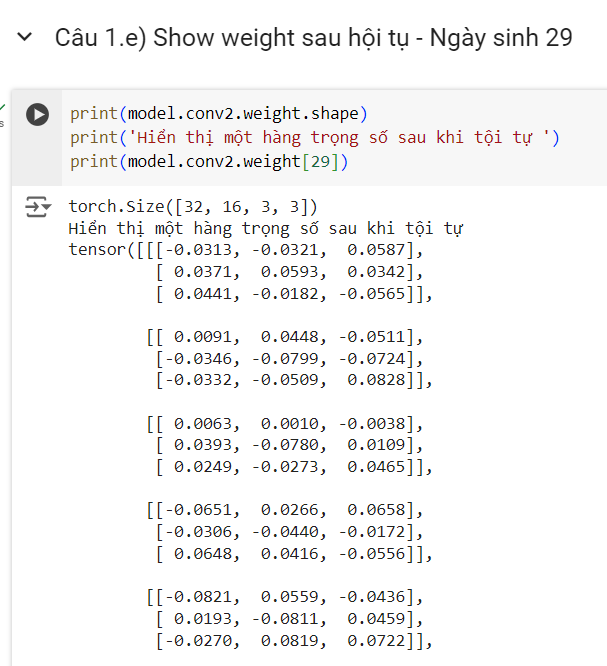
4. Tương thích với các bài toán hồi quy:

## Câu 1.e show trọng số weight

print(model.conv2.weight.shape)

print('Hiển thị một hàng trọng số sau khi tội tự ')

print(model.conv2.weight[29])



Một số trọng số khác sau hội tụ

# lớp convolution 1

print(model.conv1.weight.shape)

print(model.conv1.weight[0])

# lớp convolution 2

print(model.conv2.weight.shape)

print(model.conv2.weight[0][0])

# lớp fully connected 1

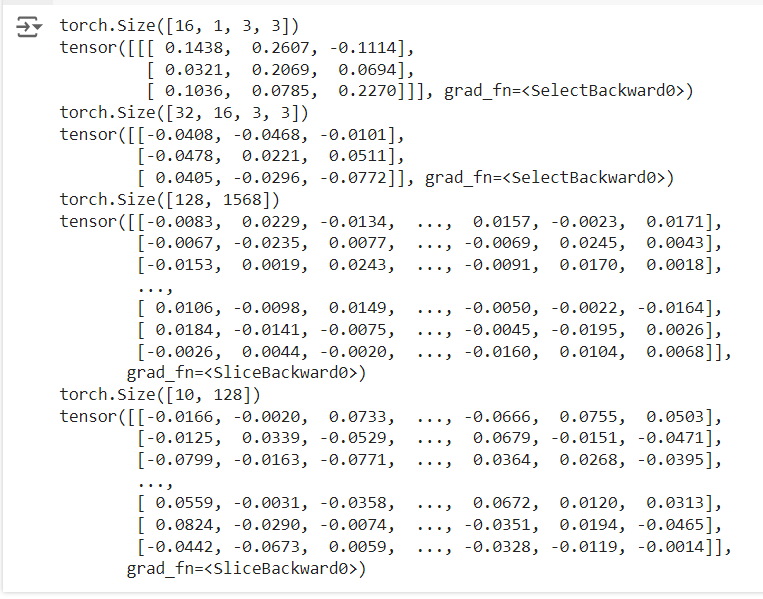
print(model.fc1.weight.shape)

print(model.fc1.weight[0:10])

# lớp fully connected 2

print(model.fc2.weight.shape)

print(model.fc2.weight[0:10])



# Câu 2 – RNN

## Câu 2.a Xây dựng model

Lựa chọn model LSTM

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

lstm\_model = Sequential()

lstm\_model.add(Embedding(vocab\_size, embedding\_dim, input\_length=max\_length))

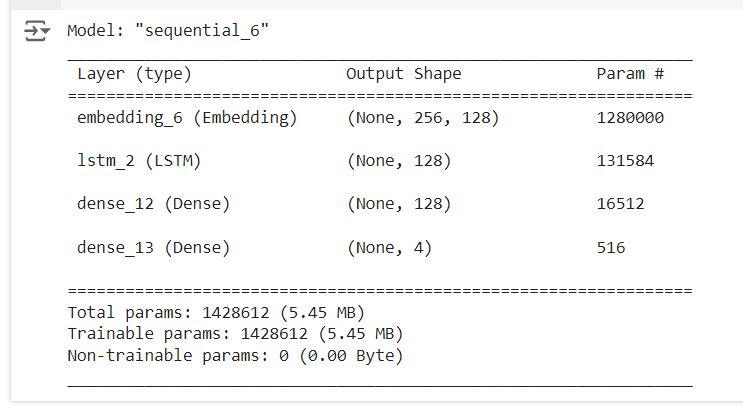
lstm\_model.add(LSTM(units))

lstm\_model.add(Dense(128, activation='relu'))

lstm\_model.add(Dense(num\_classes, activation='softmax'))

lstm\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

lstm\_model.summary()



Mô tả bộ phân loại:

- Tên lớp: Sequential

- Hàm kích hoạt:

- Dense layer 1: relu

- Dense layer 2: softmax

- Số unit đầu ra: 4

- Số LSTM cell: 128

Lý do chọn cấu trúc mô hình:

Cấu trúc này sử dụng lớp Embedding để biến đổi dữ liệu văn bản thành các vector có kích thước cố định,

giúp mô hình học được mối quan hệ ngữ nghĩa giữa các từ. Lớp LSTM giúp ghi nhớ thông tin trong chuỗi dữ liệu dài,

thích hợp cho việc xử lý văn bản. Lớp Dense với hàm kích hoạt 'relu' thêm tính phi tuyến tính vào mô hình,

và lớp Dense cuối cùng với hàm kích hoạt 'softmax' để phân loại đầu ra thành 4 lớp. Mô hình được tối ưu hóa với 'adam'

và sử dụng 'categorical\_crossentropy' cho hàm mất mát, phù hợp cho bài toán phân loại đa lớp.

## Câu 2.b Chuẩn bị dữ liệu

import pandas as pd

# Đường dẫn tới file CSV (thay đổi theo đường dẫn thực tế của bạn)

train\_csv\_path = './train.csv'

test\_csv\_path = './test.csv'

# Đọc dữ liệu từ file CSV

train\_data = pd.read\_csv(train\_csv\_path)

test\_data = pd.read\_csv(test\_csv\_path)

# Khởi tạo các danh sách để lưu dữ liệu

train\_sentences = []

test\_sentences = []

train\_labels = []

test\_labels = []

# Lưu dữ liệu từ train.csv vào các danh sách

for index, row in train\_data.iterrows():

    train\_sentences.append(row['Title'] + " " + row['Description'])

    train\_labels.append(row['Class Index'])

# Lưu dữ liệu từ test.csv vào các danh sách

for index, row in test\_data.iterrows():

    test\_sentences.append(row['Title'] + " " + row['Description'])

    test\_labels.append(row['Class Index'])

# Hiển thị kết quả để kiểm tra

print(f'Train sentences: {train\_sentences[:3]}')  # Hiển thị 3 câu đầu tiên để kiểm tra

print(f'Train labels: {train\_labels[:3]}')        # Hiển thị 3 nhãn đầu tiên để kiểm tra

print(f'Test sentences: {test\_sentences[:3]}')    # Hiển thị 3 câu đầu tiên để kiểm tra

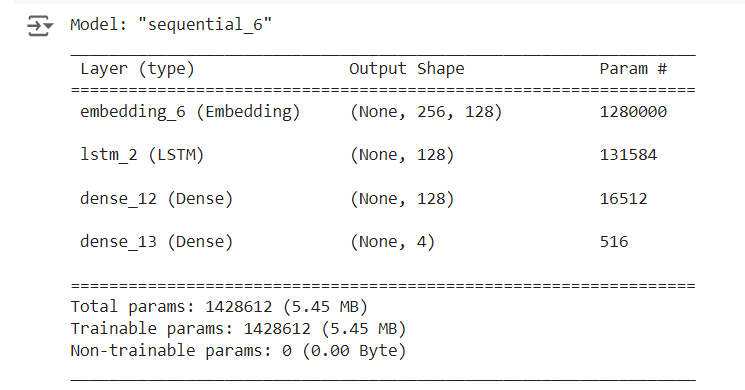
print(f'Test labels: {test\_labels[:3]}')          # Hiển thị 3 nhãn đầu tiên để kiểm tra

Dữ liệu gồm có 3 cột : Class Index,Title,Description

Chúng ta sẽ dùng cột Class Index để làm label và 2 cột Title,Description để làm dữ liệu train (cộng chúng lại)

## Câu 2.c Tính toán tham số

Ta có mô hình



vocab\_size = 10000

embedding = 64

max\_length = 256

embedding\_dim = 128  # Kích thước vector nhúng

units = 128

* input\_dim là output của lớp trước
* Embedding : Param = embedding\*vocab\_size = 128\* 10000 = 1280000
* LSTM = RNN \* 4 : Param = (units \* units + units \* input\_dim + units) \* 4 = (128 \* 128 + 128 \* 128+ 128) \* 4 = 32896\* 4 = 131584 (output của lớp trước chính là embending)
* Dense : Param = output\_dim \* input\_dim + output\_dim = 128\* 128 + 128 = 16512 (mấy lớp FC thì cũng tính giống như bên CNN)
* Dense : Param = output\_dim \* input\_dim + output\_dim = 4 \* 128 + 4 = 516
* Total Param = 1280000+ 131584 + 16512 + 516= 1428612

## Câu 2.d : Huấn luyện mô hình



## Câu 2.e Đề xuất và thực hiện 1 giải pháp để cải thiện kết quả

1. Thay đổi kích thước vector nhúng (embedding\_dim):

- Tăng hoặc giảm kích thước vector nhúng để xem cách nó ảnh hưởng đến hiệu suất mô hình. Ví dụ: `embedding\_dim = 256`.

2. Thay đổi số lượng units trong LSTM:

- Tăng hoặc giảm số lượng units trong LSTM để xem cách nó ảnh hưởng đến khả năng học và tổng quát hóa của mô hình. Ví dụ: `units = 64`.

3. Thay đổi số lượng layers của LSTM:

- Thêm hoặc giảm số lượng layers LSTM để xem cách nó ảnh hưởng đến khả năng học và tổng quát hóa. Ví dụ: `lstm\_model.add(LSTM(units, return\_sequences=True))` để thêm một layer LSTM.

4. Sử dụng các hàm activation khác cho các layers Dense:

- Thử các hàm activation khác nhau như `sigmoid` hoặc `tanh` để xem cách chúng ảnh hưởng đến hiệu suất mô hình.

5. Thay đổi optimizer và learning rate:

- Sử dụng các optimizer khác nhau như `SGD`, `RMSprop`, hoặc thậm chí tinh chỉnh learning rate để cải thiện hiệu suất mô hình.

6. Tăng kích thước vocab\_size:

- Nếu có thể, tăng kích thước của vocab\_size để mô hình có thể học được nhiều từ vựng hơn.

7. Regularization:

- Thêm các lớp Dropout để giảm overfitting, ví dụ: `lstm\_model.add(Dropout(0.2))`.

8. Thay đổi hàm loss:

- Sử dụng các hàm loss khác nhau như `binary\_crossentropy` hoặc thậm chí tùy chỉnh hàm loss để phù hợp với bài toán cụ thể.