

TRƯỜNG ĐẠI HỌC VĂN LANG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO MÔN HỌC
NHẬP MÔN XỬ LÝ ẢNH SỐ

Đề tài:

IMAGE CLASSIFICATION
USING CNN ON DATASET CIFAR 10

GVHD: Đỗ Hữu Quân,

Nguyễn Thái Anh

LHP: 243_71ITAI40803_01

Nhóm : 25

- Lê Đình Quân

- 207CT40568

TP. Hồ Chí Minh – năm 2025

LỜI CẢM ƠN

Nhóm em xin gửi lời cảm ơn sâu sắc đến Thầy Đỗ Hữu Quân và thầy Nguyễn Thái Anh, người đã tận tâm hướng dẫn, chia sẻ kiến thức và đưa ra những góp ý chân thành để đồ án của em trở nên hoàn thiện hơn. Sự nhẫn nại vào chất lượng và sự cam kết của, các thầy đã là động lực lớn giúp em vượt qua những thách thức trong quá trình nghiên cứu và thực hiện.

Cuối cùng, em xin cảm ơn hai thầy và các bạn đã luôn ủng hộ và động viên em trong suốt quá trình học tập và thực hiện đồ án này.

Mặc dù đã cố gắng hết sức nhưng chắc chắn đồ án vẫn còn thiếu sót, em rất mong nhận được góp ý từ hai thầy và các bạn để có thể cải thiện hơn trong tương lai.

Em xin chân thành cảm ơn!

TP.Hồ Chí Minh, ngày tháng năm 2025

Lê Đình Quân

Mục Lục

CHƯƠNG 1: GIỚI THIỆU	4
1.1 Lý do chọn chủ đề nghiên cứu:	4
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	5
2.1 Tổng quan về xử lý ảnh số	5
2.2 Các thuật toán và bước xử lý ảnh liên quan	6
2.3 Mô hình học sâu CNN	8
2.4 Công cụ và thư viện sử dụng	9
CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	10
3.1 Quy trình xử lý tổng thể	10
3.2 Kiến trúc hệ thống / sơ đồ luồng xử lýHệ thống được xây dựng theo kiến trúc tuần tự, nơi mỗi thành phần thực hiện một nhiệm vụ cụ thể:	17
3.3 Thiết kế các bước xử lý ảnh chi tiết:	18
3.4 Dữ liệu đầu vào / đầu ra	18
CHƯƠNG 4. CÀI ĐẶT VÀ THỰC NGHIỆM	18
4.1 Cài đặt môi trường và công cụ	18
4.2 Mã nguồn mô hình chính	18
4.3 Kết quả thực nghiệm	19
4.4 Đánh giá kết quả và so sánh	20
CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	21
5.1 Kết luận	21
5.2 Hướng phát triểnTrong tương lai, nhóm mong muốn phát triển thêm các hướng sau:21	21
CHƯƠNG 6: TÀI LIỆU THAM KHẢO	22

Tóm tắt (Abstract)

Báo cáo này trình bày chi tiết quá trình xây dựng hệ thống phân loại ảnh sử dụng mô hình học sâu Convolutional Neural Network (CNN). Dữ liệu sử dụng là CIFAR-10 – tập dữ liệu gồm 60.000 ảnh màu RGB kích thước 32x32 thuộc 10 lớp đối tượng như máy bay, ô tô, chim, chó, mèo, v.v. Dự án được thực hiện bằng ngôn ngữ Python và thư viện TensorFlow/Keras. Sau quá trình huấn luyện và đánh giá, mô hình đạt độ chính xác khoảng 80% trên tập kiểm tra. Hệ thống này được triển khai trong khuôn khổ đồ án môn "Nhập môn xử lý ảnh số", hướng đến việc áp dụng các kiến thức nền tảng về ảnh số, tiền xử lý, mạng CNN và phân loại ảnh trong thực tiễn học thuật.

Từ khóa

CNN, Deep Learning, Phân loại ảnh, CIFAR-10, Python, TensorFlow, Nhập môn xử lý ảnh số

CHƯƠNG 1: GIỚI THIỆU

1.1 Lý do chọn chủ đề nghiên cứu:

Trong thời đại số hóa ngày nay, việc xử lý và phân loại hình ảnh đã trở thành một thách thức quan trọng. Hình ảnh là một nguồn dữ liệu lớn và quan trọng, xuất phát từ nhiều lĩnh vực như y tế, công nghiệp, giáo dục và nhiều lĩnh vực khác. Tuy nhiên, việc phân loại hình ảnh một cách chính xác và hiệu quả vẫn là một thách thức đối với nhiều ứng dụng.

Dự án "Image Classification Using CNN" tập trung vào việc áp dụng mạng nơ-ron tích chập (CNN - Convolutional Neural Network) để giải quyết vấn đề phân loại hình ảnh. Mục tiêu chính là xây dựng một hệ thống có khả năng tự động học và nhận diện đối tượng trong hình ảnh với độ chính xác cao.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về xử lý ảnh số

Ảnh số là tập hợp các điểm ảnh (pixel) được mã hóa dưới dạng ma trận 2D hoặc 3D. Đối với ảnh xám (grayscale), mỗi pixel thể hiện một mức xám. Đối với ảnh màu RGB, mỗi pixel gồm 3 giá trị tương ứng với ba kênh màu Đỏ (Red), Lục (Green) và Lam (Blue). Xử lý ảnh số là một lĩnh vực của thị giác máy (computer vision), trong đó hình ảnh được biểu diễn dưới dạng ma trận số để phục vụ cho các tác vụ như nhận diện, phân loại, hoặc trích xuất thông tin.

- **Ảnh xám (grayscale):** Mỗi pixel có giá trị cường độ sáng từ 0 đến 255, đại diện cho độ đậm nhạt của màu xám.
- **Ảnh màu RGB:** Mỗi pixel gồm 3 giá trị màu cơ bản Red, Green và Blue. Các giá trị này được kết hợp để tạo nên màu sắc của pixel.

Ví dụ:

```
image = image / 255.0 # chuẩn hóa ảnh RGB về [0, 1]
```

Các thao tác thường được áp dụng trong xử lý ảnh:

- **Chuyển đổi không gian màu:** RGB \rightarrow Grayscale, HSV, YCrCb, ... để thuận lợi cho xử lý đặc trưng.
- **Cân bằng histogram:** Giúp tăng độ tương phản hình ảnh, cải thiện chất lượng đầu vào.
- **Lọc ảnh (filter):**
 - Lọc trung bình (làm mịn ảnh)
 - Lọc Gaussian (Làm mờ ảnh theo phân phối chuẩn)
 - Lọc median (Khử nhiễu muối tiêu hiệu quả)
- **Phát hiện biên (Edge Detection):** dùng các thuật toán như Sobel, Canny để tìm các ranh giới vật thể.

- **Trích xuất đặc trưng từ ảnh (Feature Extraction):** tìm các điểm đặc trưng để phục vụ cho nhận dạng và phân loại ảnh.

2.2 Các thuật toán và bước xử lý ảnh liên quan

- **Chuẩn hóa (Normalization):** Đưa giá trị pixel về phạm vi [0, 1] để giảm sai lệch gradient, giúp mô hình huấn luyện ổn định hơn.

```
# Chuẩn hóa hình ảnh
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

- Công thức chuẩn hóa : $x' = x/255$
 - ◆ Trong đó :
 - X: Giá trị gốc của pixel(0-255)
 - X':Giá trị sau chuẩn hóa(0-1)

- **Chuyển nhãn sang one-hot vector:**

Do CIFAR-10 là bài toán **phân loại 10 lớp**, nên nhãn từ dạng số (0–9) cần được chuyển sang dạng one-hot vector:

```
# One-hot encode nhãn
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)
```

Công thức được biểu diễn như sau:

$$\text{One-hot}(i)=[v_0, v_1, \dots, v_{N-1}]$$

trong đó:

$$v_j = \begin{cases} 1 & \text{neuj} = i \\ 0 & \text{neuj} \neq i \end{cases}$$

Giải thích:

- y_train và y_test: Chứa các nhãn gốc dưới dạng số nguyên, ví dụ các số từ 0 đến 9.
- to_categorical: Hàm chuyển đổi các nhãn số nguyên này thành vector one-hot.
- 10: Tham số này xác định số lượng lớp, hay độ dài của vector đầu ra. Trong trường hợp này, có 10 lớp (từ 0 đến 9).

- Tăng cường dữ liệu (Data Augmentation):

```
# Khởi tạo ImageDataGenerator với các phép biến đổi dữ liệu
datagen = ImageDataGenerator(
    rotation_range=20,           # Góc xoay ảnh trong khoảng ±20 độ
    width_shift_range=0.2,       # Dịch chuyển ngang ảnh theo chiều rộng, tỷ lệ 20%
    height_shift_range=0.2,     # Dịch chuyển ngang ảnh theo chiều cao, tỷ lệ 20%
    horizontal_flip=True        # Lật ảnh ngang
)

# Áp dụng phương pháp fit để tính toán thống kê bằng cách sử dụng dữ liệu
datagen.fit(x_train)
```

- Kỹ thuật này dùng để:

- ◆ Giảm overfitting
- ◆ Tăng khả năng tổng quát mô hình

- Một số kỹ thuật thường dùng:

- ◆ Xoay ảnh trong phạm vi ±n độ: **rotation_range**

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- ◆ Dịch ảnh ngang theo tỷ lệ phần trăm chiều rộng: **width_shift_range**

$$x' = x + \Delta x$$

- ◆ Dịch ảnh dọc theo tỷ lệ phần trăm chiều cao: **height_shift_range**

$$y' = y + \Delta y$$

- ◆ Lật ảnh theo trục ngang để tạo biến thể mới: **horizontal_flip**

$$x' = w - 1 - x$$

- **Loss function:** sử dụng categorical crossentropy cho bài toán phân loại đa lớp.
- **Optimizer:** Adam là thuật toán tối ưu phổ biến, giúp mô hình hội tụ nhanh và ổn định. Giảm overfitting bằng weight decay.

```
# Optimizer AdamW với weight decay
optimizer = optimizers.AdamW(
    learning_rate=0.001,
    weight_decay=1e-5,
    clipvalue=1.0
)
```

2.3 Mô hình học sâu CNN

Convolutional Neural Network (CNN) là mạng nơ-ron chuyên xử lý dữ liệu dạng ảnh.

Các thành phần chính của CNN gồm:

- **Conv2D** : Tầng tích chập trích xuất đặc trưng ảnh bằng bộ lọc
- **activation='relu'**: Hàm kích hoạt ReLU giúp tăng tính phi tuyến
- **padding='same'**: Giữ nguyên kích thước ảnh sau khi tích chập
- **MaxPooling2D**: Gộp đặc trưng quan trọng, giảm chiều dữ liệu
- **Dropout**: Ngăn overfitting bằng cách ngẫu nhiên bỏ bớt nơ-ron trong huấn luyện
- **Flatten** : Chuyển tensor thành vector 1 chiều để đưa vào tầng Fully Connected
- **Dense**: Tầng kết nối đầy đủ – thực hiện phân loại cuối cùng
- **Softmax**: Hàm kích hoạt phân phối xác suất cho 10 lớp

Một mô hình CNN điển hình dùng trong đề tài:

```
from tensorflow.keras import models, layers, regularizers

cnn = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

Mô hình đồ họa mô tả CNN:

Input (32x32x3)→Conv2D→MaxPooling→Dropout→Flatten→Dense→Output (10 classes)

2.4 Công cụ và thư viện sử dụng

- **Ngôn ngữ:** Python 3.10
- **Thư viện:**
 - TensorFlow, Keras: Xây dựng và huấn luyện mô hình học sâu.

```
from keras.utils import to_categorical

from keras.preprocessing.image import img_to_array, load_img
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Dense,
Activation, Flatten
import tensorflow as tf
from keras import layers, models, optimizers, losses, metrics
from keras.optimizers import Adam
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

- NumPy: Xử lý dữ liệu dạng ma trận.

```
import numpy as np
```

- Matplotlib: Hiển thị ảnh và biểu đồ.

```
import matplotlib.pyplot as plt # Hiển thị ảnh
import matplotlib.image as mpimg
```

- **Môi trường:** Google Colab, Jupyter Notebook

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ

HỆ THỐNG

3.1 Quy trình xử lý tổng thể

Quy trình xử lý của hệ thống được chia thành các bước chính như sau:

- Nạp và chuẩn bị dữ liệu (load CIFAR-10)

```
# Tải toàn bộ tập dữ liệu CIFAR-10 (gồm ảnh và nhãn)  
(X_all, y_all), _ = cifar10.load_data()
```

- Tiền xử lý ảnh (chuẩn hóa, mã hóa nhãn, chia tập train/test)

```

# Chia thủ công 80% train, 20% test
total = X_all.shape[0]
split_index = int(total * 0.8)

X_train = X_all[:split_index]
y_train = y_all[:split_index]
X_test = X_all[split_index:]
y_test = y_all[split_index:]

# ===== 4. TIỀN XỬ LÝ DỮ LIỆU =====
# Chuyển nhãn về dạng 1D
y_train = y_train.reshape(-1)
y_test = y_test.reshape(-1)

from tensorflow.keras import models, layers, optimizers, callbacks, regularizers
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Chuẩn hóa hình ảnh
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Khởi tạo ImageDataGenerator với các phép biến đổi dữ liệu
datagen = ImageDataGenerator(
    rotation_range=20,           # Góc xoay ảnh trong khoảng ±20 độ
    width_shift_range=0.2,       # Dịch chuyển ngang ảnh theo chiều rộng, tỷ lệ 20%
    height_shift_range=0.2,     # Dịch chuyển ngang ảnh theo chiều cao, tỷ lệ 20%
    horizontal_flip=True        # Lật ảnh ngang
)

# Áp dụng phương pháp fit để tính toán thống kê bằng cách sử dụng dữ liệu
datagen.fit(X_train)

# One-hot encode nhãn
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

```

- Khởi tạo mô hình CNN

```
from tensorflow.keras import models, layers, regularizers

cnn = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

```

# Dùng tf.data.Dataset để tăng tốc xử lý dữ liệu
AUTOTUNE = tf.data.AUTOTUNE
train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_ds = train_ds.shuffle(buffer_size=1000).batch(64).prefetch(AUTOTUNE)

val_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test))
val_ds = val_ds.batch(64).prefetch(AUTOTUNE)

# Optimizer AdamW với weight decay
optimizer = optimizers.AdamW(
    learning_rate=0.001,
    weight_decay=1e-5,
    clipvalue=1.0
)

# ReduceLROnPlateau và EarlyStopping callbacks
lr_scheduler = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=3,
    min_lr=1e-5
)

early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Biên dịch mô hình
cnn.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

```

- Huấn luyện mô hình

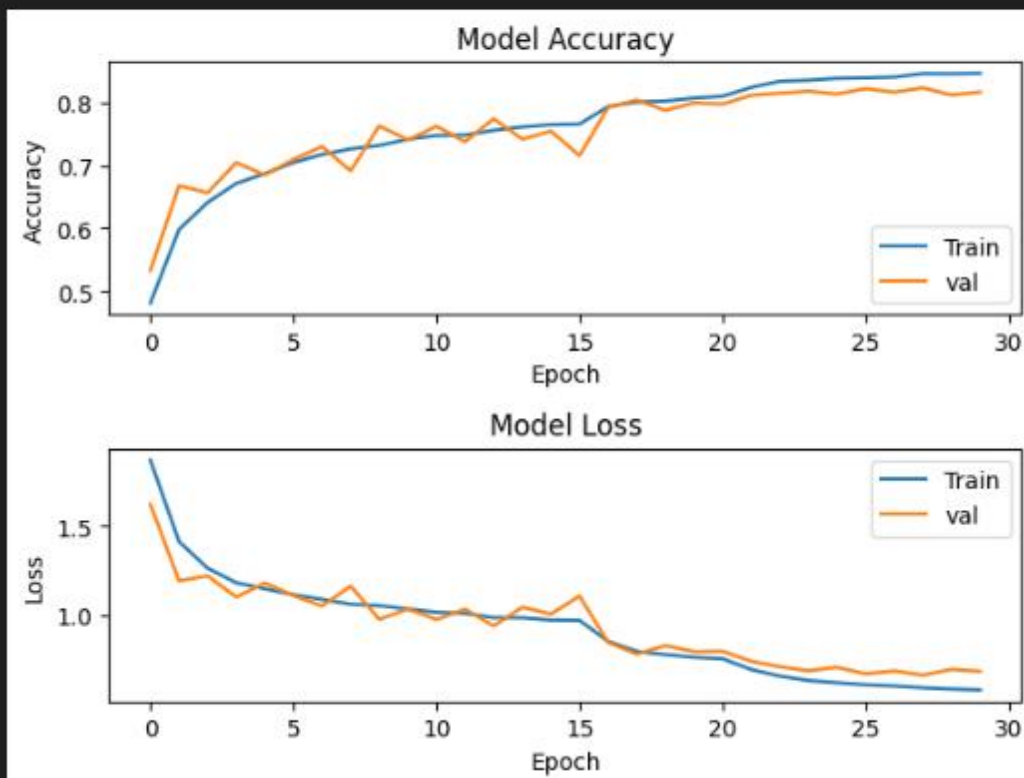
```
# Huấn luyện mô hình
history = cnn.fit(
    train_ds,
    epochs=30, # Giảm số epochs
    validation_data=val_ds, # Dùng dataset validation
    callbacks=[early_stopping, lr_scheduler] # Các callback điều chỉnh học
)

Epoch 1/30
625/625 ————— 13s 8ms/step - accuracy: 0.4291 - loss: 2.0785 - val_accuracy: 0.5330 - val_loss: 1.6189 - learning_rate: 0.0010
Epoch 2/30
625/625 ————— 4s 6ms/step - accuracy: 0.5820 - loss: 1.4750 - val_accuracy: 0.6679 - val_loss: 1.1905 - learning_rate: 0.0010
Epoch 3/30
625/625 ————— 3s 6ms/step - accuracy: 0.6350 - loss: 1.2793 - val_accuracy: 0.6567 - val_loss: 1.2178 - learning_rate: 0.0010
Epoch 4/30
625/625 ————— 4s 6ms/step - accuracy: 0.6665 - loss: 1.1847 - val_accuracy: 0.7046 - val_loss: 1.0993 - learning_rate: 0.0010
Epoch 5/30
625/625 ————— 4s 6ms/step - accuracy: 0.6870 - loss: 1.1429 - val_accuracy: 0.6850 - val_loss: 1.1783 - learning_rate: 0.0010
Epoch 6/30
625/625 ————— 3s 5ms/step - accuracy: 0.7005 - loss: 1.1161 - val_accuracy: 0.7097 - val_loss: 1.1077 - learning_rate: 0.0010
Epoch 7/30
625/625 ————— 3s 5ms/step - accuracy: 0.7186 - loss: 1.0822 - val_accuracy: 0.7304 - val_loss: 1.0492 - learning_rate: 0.0010
Epoch 8/30
625/625 ————— 3s 5ms/step - accuracy: 0.7277 - loss: 1.0525 - val_accuracy: 0.6915 - val_loss: 1.1609 - learning_rate: 0.0010
Epoch 9/30
625/625 ————— 4s 6ms/step - accuracy: 0.7313 - loss: 1.0517 - val_accuracy: 0.7632 - val_loss: 0.9745 - learning_rate: 0.0010
Epoch 10/30
625/625 ————— 5s 5ms/step - accuracy: 0.7440 - loss: 1.0317 - val_accuracy: 0.7406 - val_loss: 1.0330 - learning_rate: 0.0010
Epoch 11/30
625/625 ————— 3s 5ms/step - accuracy: 0.7513 - loss: 1.0103 - val_accuracy: 0.7625 - val_loss: 0.9736 - learning_rate: 0.0010
Epoch 12/30
625/625 ————— 4s 6ms/step - accuracy: 0.7490 - loss: 1.0056 - val_accuracy: 0.7379 - val_loss: 1.0315 - learning_rate: 0.0010
Epoch 13/30
...
Epoch 29/30
625/625 ————— 3s 5ms/step - accuracy: 0.8490 - loss: 0.5800 - val_accuracy: 0.8124 - val_loss: 0.6942 - learning_rate: 2.5000e-04
Epoch 30/30
625/625 ————— 4s 6ms/step - accuracy: 0.8467 - loss: 0.5775 - val_accuracy: 0.8169 - val_loss: 0.6829 - learning_rate: 2.5000e-04
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- Đánh giá mô hình trên tập kiểm tra

```
# Vẽ đồ thị accuracy và loss
fig = plt.figure()
plt.subplot(2, 1, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'val'], loc='lower right')

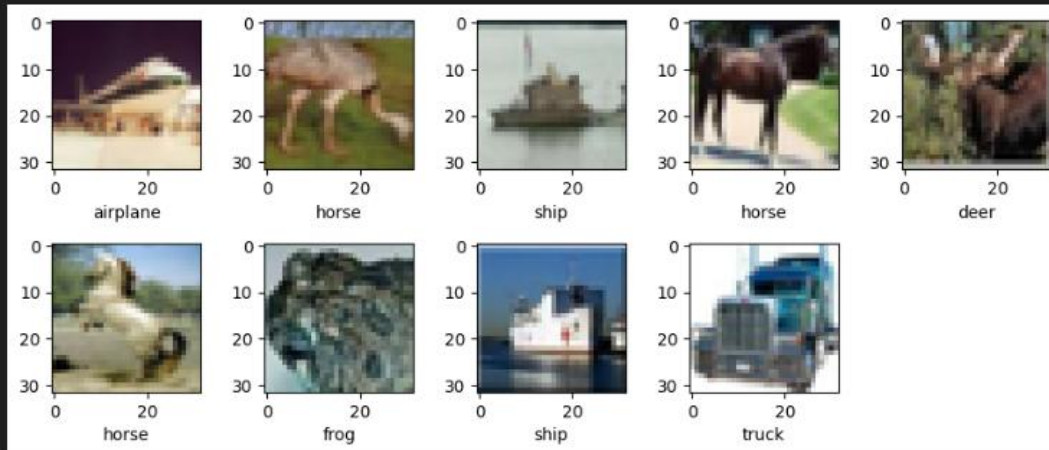
plt.subplot(2, 1, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'val'], loc='upper right')
plt.tight_layout()
plt.show()
```



- Trực quan hóa kết quả


```
# Hiển thị ngẫu nhiên 9 tấm hình trong tập test và dự đoán
#Hàm này để chọn ngẫu nhiên các tấm ảnh trong tập dữ liệu
import random
import os
predicted_classes=cnn.predict(X_test)
plt.rcParams['figure.figsize']=(9,9)
for i in range(9):
    plt.subplot(5,5,i+1)
    num=random.randint(0,len(X_test))
    plt.imshow(X_test[num])
    y_classes = [np.argmax(element) for element in predicted_classes]
    plt.xlabel(classes[y_classes[num]])
plt.tight_layout()
```

313/313 ————— 2s 3ms/step



- Cho phép người dùng tải ảnh bên ngoài và để dự đoán:



```
from google.colab import files
import io
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Hàm tải hình ảnh từ tệp được chọn và hiển thị lên giao diện người dùng
def load_and_predict():
    uploaded = files.upload()
    for filename in uploaded.keys():
        img_test = load_image(io.BytesIO(uploaded[filename]))
        plt.figure(figsize=(3, 3)) # Đặt kích thước hình ảnh hiển thị
        plt.imshow(img_test[0])
        plt.axis('off')
        plt.show()
        ob = cnn.predict(img_test)
        prediction = np.argmax(ob)
        confidence = np.max(ob) * 100 # Tính xác suất dự đoán chính xác
        print(f'Dự đoán hình ảnh: {classes[prediction]} với độ chính xác: {confidence:.2f}%')

# Hàm để load và chuẩn bị hình ảnh
def load_image(image_bytes):
    image = Image.open(image_bytes)
    image = image.resize((32, 32))
    image = np.expand_dims(image, axis=0)
    image = np.array(image)
    img = image.astype('float32')
    img = img / 255.0
    return img

# Biến classes
classes = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

# Tạo nút để chọn hình ảnh và dự đoán
print("Chọn 1 hình ảnh để dự đoán:")
load_and_predict()

Chọn 1 hình ảnh để dự đoán:
[Choose Files] No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving tải xuống (7).jpg to tải xuống (7) (1).jpg

1/1 ————— 0s 407ms/step
Dự đoán hình ảnh: frog với độ chính xác: 91.10%
```

3.2 Kiến trúc hệ thống / sơ đồ luồng xử lý

Hệ thống được xây dựng theo kiến trúc tuần tự, nơi mỗi thành phần thực hiện một nhiệm vụ cụ thể:

- Khối dữ liệu: tải CIFAR-10 từ thư viện Keras.
- Khối xử lý: chuẩn hóa dữ liệu ảnh RGB và one-hot encode nhãn.
- Khối mô hình: xây dựng CNN với các tầng Conv, Pooling, Flatten, Dense.
- Khối huấn luyện: tối ưu bằng Adam, dùng loss categorical_crossentropy.
- Khối đánh giá: kiểm tra accuracy, trực quan hóa loss/accuracy qua epoch.

3.3 Thiết kế các bước xử lý ảnh chi tiết:

- Ảnh đầu vào được resize về (32x32x3) và chuẩn hóa về khoảng [0,1].
- Nhận được chuyển từ số nguyên sang dạng one-hot vector.
- Không thực hiện thêm các bộ lọc không gian (vì mô hình CNN đã tự học đặc trưng).
- Augmentation có thể bao gồm: xoay, dịch trái/phải, lật ngang ảnh.

3.4 Dữ liệu đầu vào / đầu ra

- Đầu vào: ảnh màu RGB kích thước 32x32, mỗi ảnh thuộc một trong 10 lớp (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
- Đầu ra: vector xác suất 10 chiều (softmax), nhãn có xác suất cao nhất sẽ được chọn làm kết quả phân loại.

CHƯƠNG 4. CÀI ĐẶT VÀ THỰC NGHIỆM

4.1 Cài đặt môi trường và công cụ

- Python 3.10
- TensorFlow 2.x, Keras, NumPy, Matplotlib
- Chạy trên Google Colab với GPU hỗ trợ

4.2 Mã nguồn mô hình chính

- Dữ liệu đầu vào là dataset được tải trực tiếp từ thư viện keras

```
# Tải toàn bộ tập dữ liệu CIFAR-10 (gồm ảnh và nhãn)
(x_all, y_all), _ = cifar10.load_data()
```

- **Mô hình CNN được sử dụng**

```
from tensorflow.keras import models, layers, regularizers

cnn = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

4.3 Kết quả thực nghiệm

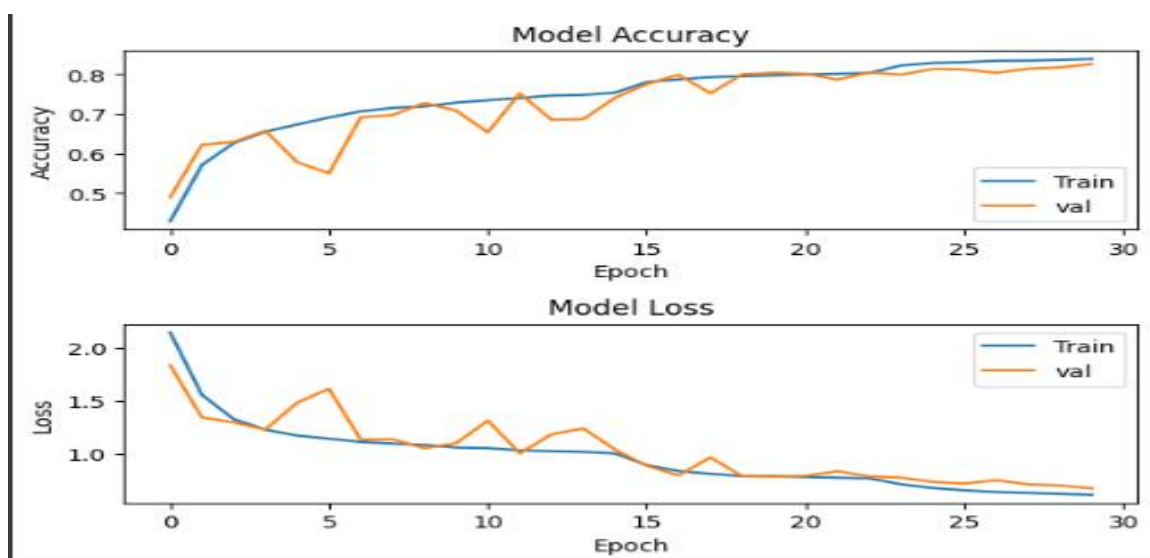
- Mô hình đạt độ chính xác khoảng 80% trên tập test sau 20 epoch.
- Quan sát loss giảm đều và accuracy tăng theo từng epoch.
- Không xảy ra overfitting rõ rệt.
- **Kết quả đạt được dựa trên số epoch sau khi huấn luyện mô hình:**

```

Epoch 1/30
625/625 — 13s 8ms/step - accuracy: 0.4291 - loss: 2.0785 - val_accuracy: 0.5330 - val_loss: 1.6189 - learning_rate: 0.0010
Epoch 2/30
625/625 — 4s 6ms/step - accuracy: 0.5820 - loss: 1.4750 - val_accuracy: 0.6679 - val_loss: 1.1905 - learning_rate: 0.0010
Epoch 3/30
625/625 — 3s 6ms/step - accuracy: 0.6350 - loss: 1.2793 - val_accuracy: 0.6567 - val_loss: 1.2178 - learning_rate: 0.0010
Epoch 4/30
625/625 — 4s 6ms/step - accuracy: 0.6665 - loss: 1.1847 - val_accuracy: 0.7046 - val_loss: 1.0993 - learning_rate: 0.0010
Epoch 5/30
625/625 — 4s 6ms/step - accuracy: 0.6870 - loss: 1.1429 - val_accuracy: 0.6850 - val_loss: 1.1783 - learning_rate: 0.0010
Epoch 6/30
625/625 — 3s 5ms/step - accuracy: 0.7005 - loss: 1.1161 - val_accuracy: 0.7097 - val_loss: 1.1077 - learning_rate: 0.0010
Epoch 7/30
625/625 — 3s 5ms/step - accuracy: 0.7186 - loss: 1.0822 - val_accuracy: 0.7304 - val_loss: 1.0492 - learning_rate: 0.0010
Epoch 8/30
625/625 — 3s 5ms/step - accuracy: 0.7277 - loss: 1.0525 - val_accuracy: 0.6915 - val_loss: 1.1609 - learning_rate: 0.0010
Epoch 9/30
625/625 — 4s 6ms/step - accuracy: 0.7313 - loss: 1.0517 - val_accuracy: 0.7632 - val_loss: 0.9745 - learning_rate: 0.0010
Epoch 10/30
625/625 — 5s 5ms/step - accuracy: 0.7440 - loss: 1.0317 - val_accuracy: 0.7406 - val_loss: 1.0330 - learning_rate: 0.0010
Epoch 11/30
625/625 — 3s 5ms/step - accuracy: 0.7513 - loss: 1.0103 - val_accuracy: 0.7625 - val_loss: 0.9736 - learning_rate: 0.0010
Epoch 12/30
625/625 — 4s 6ms/step - accuracy: 0.7490 - loss: 1.0056 - val_accuracy: 0.7379 - val_loss: 1.0315 - learning_rate: 0.0010
Epoch 13/30
...
Epoch 29/30
625/625 — 3s 5ms/step - accuracy: 0.8490 - loss: 0.5800 - val_accuracy: 0.8124 - val_loss: 0.6942 - learning_rate: 2.5000e-04
Epoch 30/30
625/625 — 4s 6ms/step - accuracy: 0.8467 - loss: 0.5775 - val_accuracy: 0.8169 - val_loss: 0.6829 - learning_rate: 2.5000e-04

```

● Biểu đồ Accuracy sau khi huấn luyện mô hình :



4.4 Đánh giá kết quả và so sánh

- Kết quả mô hình đạt yêu cầu cơ bản môn học.
- Độ chính xác có thể tăng nếu sử dụng kỹ thuật data augmentation hoặc training nhiều epoch hơn.
- So với baseline (random guess: 10%), kết quả này vượt trội.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG

PHÁT TRIỂN

5.1 Kết luận

Trong đồ án này, nhóm đã xây dựng thành công mô hình phân loại ảnh sử dụng mạng nơ-ron tích chập (CNN) trên tập dữ liệu CIFAR-10. Kết quả mô hình đạt độ chính xác khoảng 80%, thể hiện khả năng học và phân loại ảnh tương đối tốt. Mô hình được triển khai bằng Python, sử dụng các thư viện TensorFlow/Keras và thực hiện huấn luyện trên nền tảng Google Colab.

Việc áp dụng kiến thức môn học "Nhập môn xử lý ảnh số" vào bài toán thực tiễn giúp nhóm hiểu rõ hơn về quy trình xử lý ảnh, tiền xử lý dữ liệu, thiết kế mạng học sâu và cách đánh giá mô hình. Đây là nền tảng vững chắc cho các nghiên cứu và ứng dụng nâng cao hơn trong tương lai.

5.2 Hướng phát triển

Trong tương lai, nhóm mong muốn phát triển thêm các hướng sau:

Tăng độ chính xác mô hình: thử nghiệm với các kiến trúc CNN phức tạp hơn như ResNet, VGG hoặc sử dụng kỹ thuật Transfer Learning.

Ứng dụng kỹ thuật tăng cường dữ liệu (data augmentation) để cải thiện khả năng tổng quát hóa của mô hình.

Tối ưu mô hình: sử dụng kỹ thuật regularization, dropout hoặc thay đổi optimizer.

Triển khai mô hình: xây dựng giao diện đơn giản để người dùng có thể tải ảnh và xem kết quả phân loại trực tiếp.

So sánh mô hình: thử nghiệm nhiều mô hình khác nhau để đánh giá và chọn ra mô hình tối ưu nhất cho tập dữ liệu CIFAR-10.

<https://github.com/Ldinhquan/-n-cu-i-k---Nh-p-m-n-x-l-nh-s-/blob/main/Doancuoiky.ipynb>

CHƯƠNG 6: TÀI LIỆU THAM KHẢO

[1] Image classification using CNN – 88%

<https://www.kaggle.com/code/faressayah/cifar-10-images-classification-using-cnns-88>

[2] Image Classification on CIFAR-10 - Papers With Code

<https://paperswithcode.com/sota/image-classification-on-cifar-10>

[3] Chat GPT

<https://chatgpt.com/>

[4] Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images* [Technical Report]. University of Toronto. (Bộ dữ liệu CIFAR-10)

[5] Keras Documentation. *Image classification with modern convnets*. [Online] Available:

https://keras.io/examples/vision/image_classification_from_scratch/