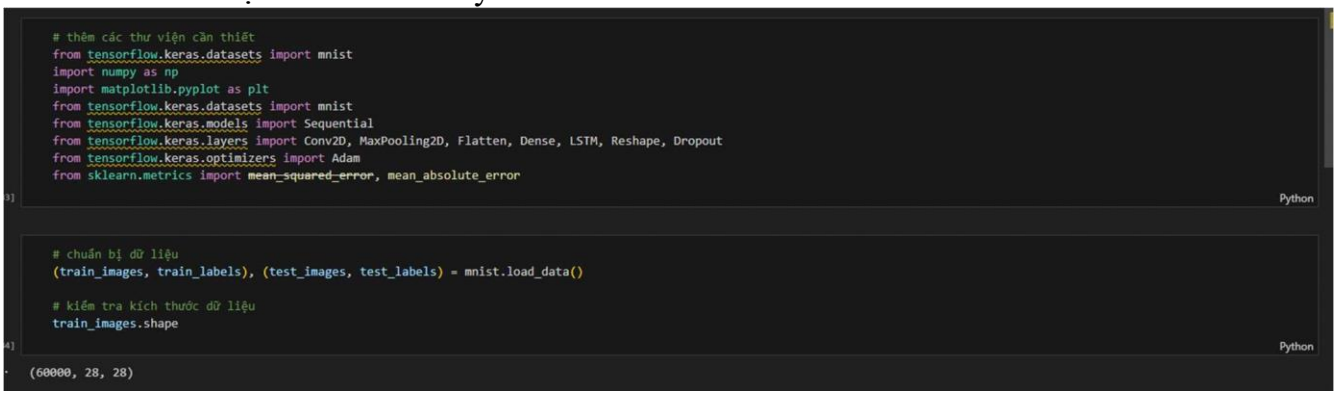
Bài tập intsyst01 - Nguyễn Quang Thắng - B21DCCN669

Bài 1: Dữ liệu MNIST

- Tạo các mô hình và Training

Thêm các thư viện cần thiết và lấy data



```
# thêm các thư viện cần thiết
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, LSTM, Reshape, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

Python

```
# chuẩn bị dữ liệu
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# kiểm tra kích thước dữ liệu
train_images.shape
```

Python

```
(60000, 28, 28)
```

Trong các thư viện trên thì numpy là thư viện được sử dụng để xử lý các dữ liệu đầu vào, đổi từ dữ liệu ảnh đầu vào thành một mảng numpy

Matplot để vẽ các biểu đồ so sánh giữa 2 mô hình CNN và

RNN Tensorflow và keras dùng để tạo và huấn luyện các mô

hình CNN và RNN Tiền xử lý dữ liệu

```
# Tiền xử lý dữ liệu
train_images=  train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

Hai dòng trên thay đổi hình dạng (shape) của mảng train_images và test_images từ định dạng 2D (28, 28) thành định dạng 4D (số lượng mẫu, chiều cao, chiều rộng, số kênh màu).

Giúp cho các giá trị trong mảng được chuẩn hóa về giá trị từ 0-1.

- Xây dựng Model CNN

```python
#2. Xây dựng mô hình CNN
def create_cnn_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation = 'relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation = 'relu'))
    model.add(Flatten())
    model.add(Dense(64, activation = 'relu'))
    model.add(Dense(10, activation ='softmax'))
    model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

Trong đó thì có các bước lần lượt là:

- model = Sequential() dùng để khởi tạo mô hình tuần tự
- model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))) Lớp tích chập đầu tiên với 32 bộ lọc với kích thước 3x3. Sử dụng hàm kích hoạt ReLU.

input_shape cho biết kích thước đầu vào: (28, 28, 1) cho hình ảnh MNIST

- model.add(MaxPooling2D((2, 2)))

Lớp giảm kích thước (pooling). Sử dụng pooling 2x2.

- model.add(Conv2D(64, (3, 3), activation='relu'))

Lớp tích chập thứ hai có 64 bộ lọc với kích thước 3x3 và sử dụng hàm kích hoạt ReLU

- model.add(MaxPooli

ng2D((2, 2))) Lớp giảm

kích thước (pooling) thứ

hai

- model.add(Conv2D(64, (3, 3), activation='relu'))

Lớp tích chập thứ ba với 64 bộ lọc với kích thước 3x3

- model.add(Dense(64, activation='relu'))

Lớp Dense thứ nhất với 64 nơ-ron với hàm kích hoạt ReLU

- model.add(Dense(10, activation='softmax'))

Lớp Dense đầu ra với 10 nơ-ron tương ứng với 10 lớp (các chữ số từ 0 đến 9)

- model.add(Dense(10, activation='softmax'))

Sử dụng hàm kích hoạt softmax để đưa ra xác suất cho mỗi lớp

- Xây dựng Model RNN

```python
# 3. Xây dựng mô hình RNN với ít nhất 5 Lớp
def create_rnn_model():
    model = Sequential()

    # Chuyển đổi hình ảnh 28x28 thành chuỗi
    model.add(Reshape((28, 28), input_shape=(28, 28, 1)))

    # LSTM layer 1
    model.add(LSTM(64, return_sequences=True))
    model.add(Dropout(0.2))

    # LSTM layer 2
    model.add(LSTM(64, return_sequences=True))
    model.add(Dropout(0.2))

    # LSTM layer 3
    model.add(LSTM(64, return_sequences=True))
    model.add(Dropout(0.2))

    # LSTM layer 4
    model.add(LSTM(64))
    model.add(Dropout(0.2))

    # Dense layer
    model.add(Dense(64, activation='relu'))

    # Output layer
    model.add(Dense(10, activation='softmax'))

    # Compile model
    model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model
```

- model = Sequential() # Khởi tạo mô hình tuần tự
- model.add(Reshape((28, 28), input_shape=(28, 28, 1))) để chuyển đổi hình ảnh từ (28,

28, 1) sang (28, 28) để xử lý như chuỗi

- model.add(LSTM(64,

return_sequences=True)) Lớp LSTM

đầu tiên với 64 đơn vị, trả về chuỗi

- model.add(Dropout(0.2))

Lớp Dropout để giảm overfitting bằng cách loại bỏ 20% đơn vị

- model.add(LSTM(64, return_sequences=True))
- model.add(Dropout(0.2))
- model.add(LSTM(64, return_sequences=True))
- model.add(Dropout(0.2))
- model.add(LSTM(64))
- model.add(Dense(64, activation='relu'))

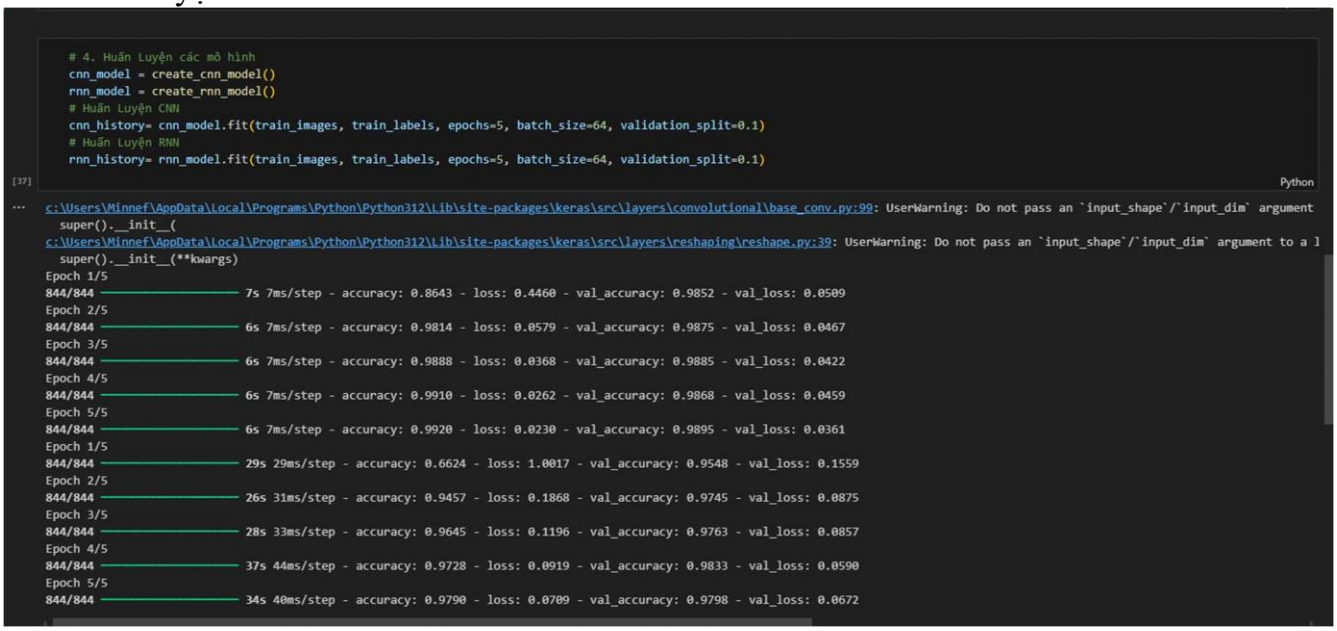
Lớp Dense với 64 đơn vị và hàm kích hoạt ReLU

- model.add(Dense(10,

activation='softmax')) Lớp đầu ra

với 10 đơn vị và hàm softmax

- Huấn luyện các mô hình



```
# 4. Huấn Luyện các mô hình
cnn_model = create_cnn_model()
rnn_model = create_rnn_model()
# Huấn Luyện CNN
cnn_history= cnn_model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_split=0.1)
# Huấn Luyện RNN
rnn_history= rnn_model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_split=0.1)
```

[37] Python

```
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\reshaping\reshape.py:39: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a l
  super().__init__(**kwargs)
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.8643 - loss: 0.4460 - val_accuracy: 0.9852 - val_loss: 0.0509
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9814 - loss: 0.0579 - val_accuracy: 0.9875 - val_loss: 0.0467
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9888 - loss: 0.0368 - val_accuracy: 0.9885 - val_loss: 0.0422
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9910 - loss: 0.0262 - val_accuracy: 0.9868 - val_loss: 0.0459
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9920 - loss: 0.0230 - val_accuracy: 0.9895 - val_loss: 0.0361
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 29s 29ms/step - accuracy: 0.6624 - loss: 1.0017 - val_accuracy: 0.9548 - val_loss: 0.1559
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 26s 31ms/step - accuracy: 0.9457 - loss: 0.1868 - val_accuracy: 0.9745 - val_loss: 0.0875
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 28s 33ms/step - accuracy: 0.9645 - loss: 0.1196 - val_accuracy: 0.9763 - val_loss: 0.0857
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 37s 44ms/step - accuracy: 0.9728 - loss: 0.0919 - val_accuracy: 0.9833 - val_loss: 0.0590
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 34s 40ms/step - accuracy: 0.9790 - loss: 0.0709 - val_accuracy: 0.9798 - val_loss: 0.0672
```

Huấn luyện mô hình với epoch là 5 và batch_size là 64

- Đánh giá mô hình

```python
# 5. Đánh giá mô hình
cnn_predictions = cnn_model.predict(test_images)
rnn_predictions = rnn_model.predict(test_images)
cnn_predictions = np.argmax(cnn_predictions, axis=1)
rnn_predictions = np.argmax(rnn_predictions, axis=1)
cnn_mse = mean_squared_error(test_labels, cnn_predictions)
cnn_mae = mean_absolute_error(test_labels, cnn_predictions)
cnn_rmse = np.sqrt(cnn_mse)
rnn_mse = mean_squared_error(test_labels, rnn_predictions)
rnn_mae = mean_absolute_error(test_labels, rnn_predictions)
```

```
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\reshaping\reshape.py:39: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a l
  super().__init__(**kwargs)
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.8643 - loss: 0.4460 - val_accuracy: 0.9852 - val_loss: 0.0509
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9814 - loss: 0.0579 - val_accuracy: 0.9875 - val_loss: 0.0467
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9888 - loss: 0.0368 - val_accuracy: 0.9885 - val_loss: 0.0422
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9910 - loss: 0.0262 - val_accuracy: 0.9868 - val_loss: 0.0459
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9920 - loss: 0.0230 - val_accuracy: 0.9895 - val_loss: 0.0361
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 29s 29ms/step - accuracy: 0.6624 - loss: 1.0017 - val_accuracy: 0.9548 - val_loss: 0.1559
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 26s 31ms/step - accuracy: 0.9457 - loss: 0.1868 - val_accuracy: 0.9745 - val_loss: 0.0875
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 28s 33ms/step - accuracy: 0.9645 - loss: 0.1196 - val_accuracy: 0.9763 - val_loss: 0.0857
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 37s 44ms/step - accuracy: 0.9728 - loss: 0.0919 - val_accuracy: 0.9833 - val_loss: 0.0590
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 34s 40ms/step - accuracy: 0.9790 - loss: 0.0709 - val_accuracy: 0.9798 - val_loss: 0.0672
```

Đánh giá mô hình qua các chỉ số là mse,

mae, rmse Kết quả train mô hình

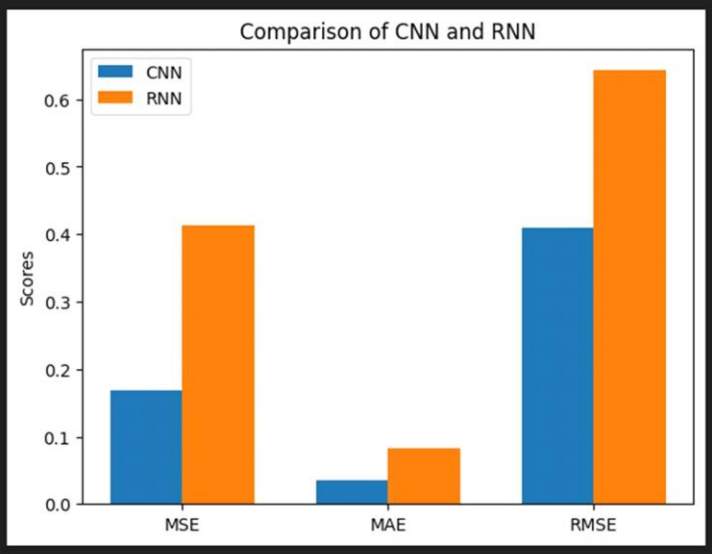
- Vẽ các biểu đồ so sánh

```python
# 6. Vẽ biểu đồ so sánh
labels = ['MSE', 'MAE', 'RMSE']
cnn_scores = [cnn_mse, cnn_mae, cnn_rmse]
rnn_scores = [rnn_mse, rnn_mae, rnn_rmse]
x = np.arange(len(labels)) # the label locations
width = 0.35 # the width of the bars
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, cnn_scores, width, label='CNN')
rects2 = ax.bar(x + width/2, rnn_scores, width, label='RNN')
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Comparison of CNN and RNN')
ax.set_xticks(x)
ax.set_xticklabels (labels)
ax.legend()
# Show the plot
plt.show()
```

- Kết quả của các biểu đồ

```
ax.set_xticklabels (labels)
ax.legend()
# Show the plot
plt.show()
```

[39]



Bài 2: Dữ liệu IMDB

- Thêm các thư viện cần thiết và tiền xử lý dữ liệu

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data( num_words=10000)
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense, LSTM, Dropout
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score, mean_squared_error, mean_absolute_error
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Tiền xử lý dữ liệu: Padding
maxlen = 300 # Độ dài tối đa của mỗi câu =
train_data = pad_sequences (train_data, maxlen=maxlen)
test_data = pad_sequences (test_data, maxlen=maxlen)
```

- Xây dựng mô hình CNN

```python
# 2. Xây dựng mô hình CNN với 5 lớp
def create_cnn_model():
    model = Sequential()

    # Lớp Embedding để biểu diễn văn bản
    model.add(Embedding(input_dim=5000, output_dim=128, input_length=maxlen))

    # Lớp Conv1D 1
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))

    # Lớp Conv1D 2
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))

    # Lớp Conv1D 3
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))

    # Lớp Conv1D 4
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))

    # Lớp Flatten
    model.add(Flatten())

    # Lớp Dense với 128 đơn vị
    model.add(Dense(128, activation='relu'))

    # Lớp Dropout để giảm overfitting
    model.add(Dropout(0.5))

    # Lớp đầu ra với hàm kích hoạt sigmoid
    model.add(Dense(1, activation='sigmoid'))

    # Compile mô hình
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

- model = Sequential()

Khởi tạo mô hình tuần tự (sequential model)

- model.add(Embedding(10000, 128, input_length=maxlen))
    - 10000: kích thước từ điển
    - 128: kích thước vector nhúng cho mỗi từ
    - input_length=maxlen: chiều dài của các chuỗi đầu vào
- model.add(Conv1D(128, 5, activation='relu'))

Lớp tích chập 1D đầu tiên

- 128: số lượng bộ lọc (filters) sẽ được sử dụng.
- 5: kích thước kernel (số từ mà bộ lọc sẽ xem xét cùng một lúc).
- activation='relu': sử dụng hàm kích hoạt ReLU.

- model.add(MaxPooling1D(

pool_size=2)) Lớp pooling 1D

- pool_size=2: giảm chiều dài đầu ra của lớp tích chập bằng cách lấy max của mỗi cặp giá trị.

- model.add(Conv1D(128, 5,

activation='relu')) Lớp tích chập 1D

thứ hai

- model.add(MaxPooling1D(pool_size=2))
- model.add(Flatten())

Chuyển đổi đầu ra 2D của lớp pooling thành 1D để có thể kết nối với các lớp Dense.

- model.add(Dense(128,

activation='relu')) Lớp Dense

- 128: số lượng đơn vị trong lớp Dense.
- activation='relu': sử dụng hàm kích hoạt ReLU.

- model.add(

Dropout(0.5))

Lớp Dropout

- Dropout 50% đơn vị trong lớp trước đó để giảm overfitting.

- model.add(Dense(1, activation='sigmoid'))

- 1: chỉ có một đơn vị đầu ra cho nhiệm vụ phân loại nhị phân.
- activation='sigmoid': hàm kích hoạt sigmoid được sử dụng để dự đoán xác suất

- Xây dựng mô hình RNN

```python
# 3. Xây dựng mô hình RNN
def create_rnn_model():
    model = Sequential()

    # Lớp Embedding để biểu diễn văn bản
    model.add(Embedding(input_dim=10000, output_dim=128, input_length=maxlen))

    # Lớp LSTM 1
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.5))

    # Lớp LSTM 2
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.5))

    # Lớp LSTM 3
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.5))

    # LSTM cuối cùng (lớp LSTM 4)
    model.add(LSTM(128))

    # Lớp Dense ẩn với 128 đơn vị
    model.add(Dense(128, activation='relu'))

    # Lớp đầu ra cho phân loại nhị phân
    model.add(Dense(1, activation='sigmoid'))

    # Compile mô hình
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

- Huấn luyện các mô hình

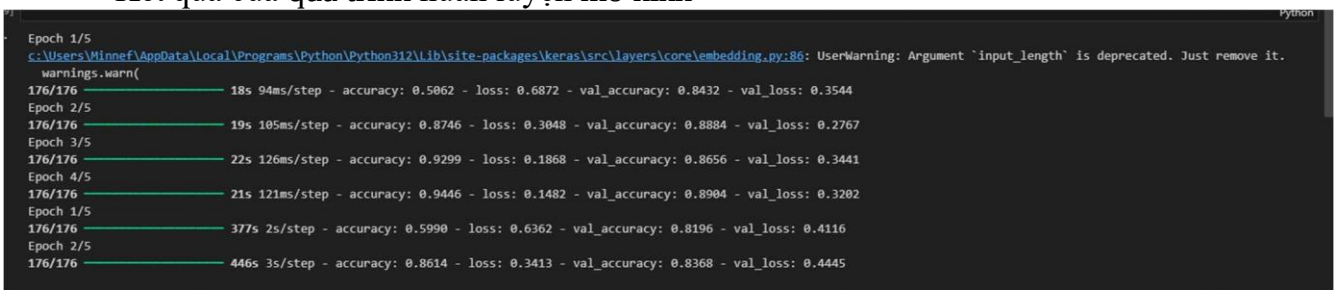
```python
# 4. Huấn Luyện các mô hình
cnn_model = create_cnn_model()
rnn_model = create_rnn_model()
early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
# Huấn Luyện CNN
cnn_history= cnn_model.fit(train_data, train_labels, epochs=5, batch_size=128, validation_split=0.1, callbacks=[early_stopping])
# Huấn Luyện RNN
rnn_history= rnn_model.fit(train_data, train_labels, epochs=5, batch_size=128, validation_split=0.1, callbacks=[early_stopping])
```

Python

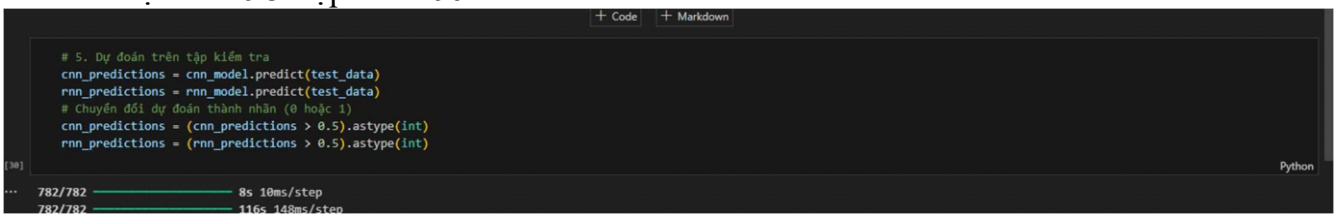
```
Epoch 1/5
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\embedding.py:86: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
176/176 ━━━━━━━━━━━━━━━━━━━━ 18s 94ms/step - accuracy: 0.5062 - loss: 0.6872 - val_accuracy: 0.8432 - val_loss: 0.3544
Epoch 2/5
176/176 ━━━━━━━━━━━━━━━━━━━━ 19s 105ms/step - accuracy: 0.8746 - loss: 0.3048 - val_accuracy: 0.8884 - val_loss: 0.2767
Epoch 3/5
176/176 ━━━━━━━━━━━━━━━━━━━━ 22s 126ms/step - accuracy: 0.9299 - loss: 0.1868 - val_accuracy: 0.8656 - val_loss: 0.3441
Epoch 4/5
176/176 ━━━━━━━━━━━━━━━━━━━━ 21s 121ms/step - accuracy: 0.9446 - loss: 0.1482 - val_accuracy: 0.8904 - val_loss: 0.3202
Epoch 1/5
176/176 ━━━━━━━━━━━━━━━━━━━━ 377s 2s/step - accuracy: 0.5990 - loss: 0.6362 - val_accuracy: 0.8196 - val_loss: 0.4116
Epoch 2/5
176/176 ━━━━━━━━━━━━━━━━━━━━ 446s 3s/step - accuracy: 0.8614 - loss: 0.3413 - val_accuracy: 0.8368 - val_loss: 0.4445
```

Train 5 epoch với batch size là 128

- Kết quả của quá trình huấn luyện mô hình

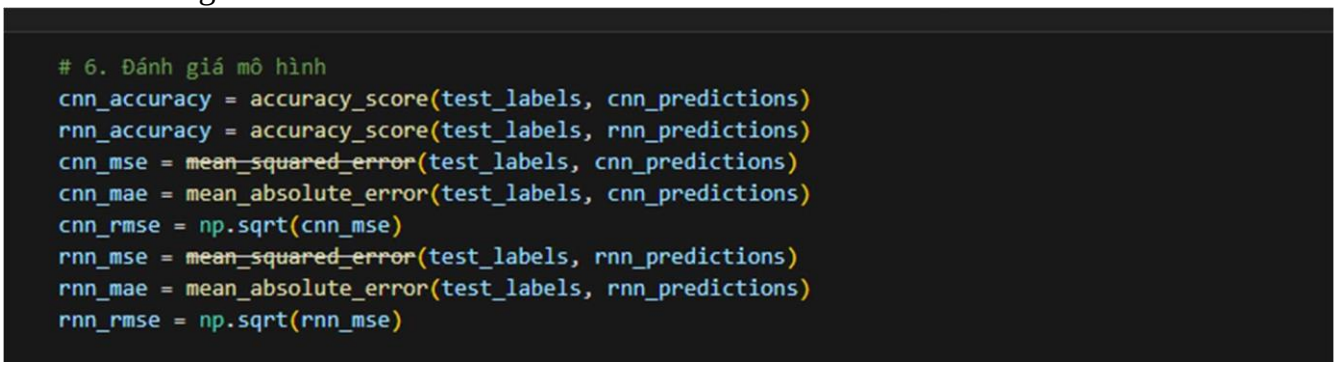


- Dự đoán trên tập kiểm tra



Dự đoán của mô hình ra số từ 0-1, nên chuyển đổi thành 2 giá trị là 0 và 1 để phân loại.

- Đánh giá mô hình

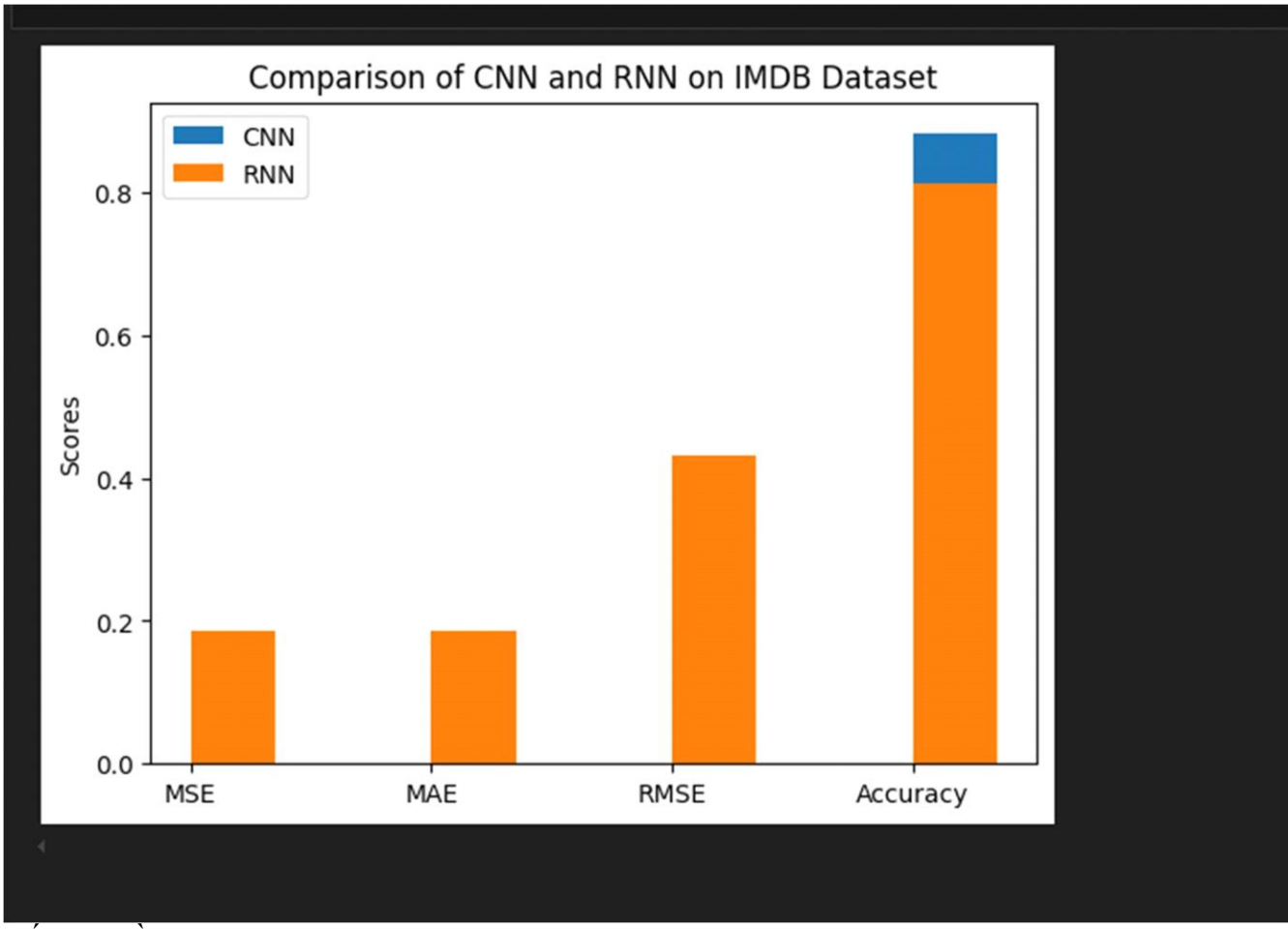


```
# 6. Đánh giá mô hình
cnn_accuracy = accuracy_score(test_labels, cnn_predictions)
rnn_accuracy = accuracy_score(test_labels, rnn_predictions)
cnn_mse = mean_squared_error(test_labels, cnn_predictions)
cnn_mae = mean_absolute_error(test_labels, cnn_predictions)
cnn_rmse = np.sqrt(cnn_mse)
rnn_mse = mean_squared_error(test_labels, rnn_predictions)
rnn_mae = mean_absolute_error(test_labels, rnn_predictions)
rnn_rmse = np.sqrt(rnn_mse)
```

Đánh giá mô hình thông qua các tham số là mse, mae, rmse

- Vẽ biểu đồ so sánh

```python
# 7. Vẽ biểu đồ so sánh
labels = ['MSE', 'MAE', 'RMSE', 'Accuracy']
cnn_scores = [cnn_mse, cnn_mae, cnn_rmse, cnn_accuracy]
rnn_scores = [rnn_mse, rnn_mae, rnn_rmse, rnn_accuracy]
x = np.arange(len(labels)) # Vị trí nhãn
width = 0.35 # Độ rộng của các thanh
fig, ax = plt.subplots()
rects1 = ax.bar(x + width/2, cnn_scores, width, label='CNN')
rects2 = ax.bar(x + width/2, rnn_scores, width, label='RNN')
# Thêm tiêu đề, nhãn và chú thích
ax.set_ylabel('Scores')
ax.set_title('Comparison of CNN and RNN on IMDB Dataset')
ax.set_xticks(x)
ax.set_xticklabels (labels)
ax.legend()
# Hiển thị biểu đồ
plt.show()
```

- 

Kết quả đồ thị

Bài 3: Dữ liệu Boston_housing

```
from tensorflow.keras.datasets import boston_housing
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import boston_housing
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Conv1D, MaxPooling1D, Flatten, BatchNormalization
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from tensorflow.keras.callbacks import EarlyStopping
```

+ Code + Markdown

```
# Tiền xử lý dữ liệu: chuẩn hóa
(train_data, train_targets), (test_data, test_targets) = (boston_housing.load_data())
scaler = StandardScaler()
train_data = scaler.fit_transform(train_data)
test_data = scaler.transform(test_data)
```

```
# Reshape dữ liệu cho mô hình CNN và RNN
train_data = train_data.reshape((train_data.shape[0], train_data.shape[1], 1))
test_data = test_data.reshape((test_data.shape[0], test_data.shape[1], 1))
```

- Chuẩn bị dữ liệu và them các thư viện
- Xây dựng mô hình CNN

```
# 2.Xây dựng mô hình CNN với 5 lớp
def create_cnn_model():
    model = Sequential()
    model.add(Conv1D (64, 2, activation='relu', input_shape=(train_data.shape[1], 1)))
    model.add(BatchNormalization())
    model.add(MaxPooling1D (pool_size=2))
    model.add(Conv1D(128, 2, activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(256, 2, activation='relu'))
    model.add(Flatten())
    model.add(Dense (128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1)) # Đầu ra cho hồi quy
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
    return model
```

- Xây dựng mô hình RNN

```python
# 3. Xây dựng mô hình RNN với 5 Lớp
def create_rnn_model():
    model = Sequential()
    model.add(LSTM (64, return_sequences=True, input_shape=(train_data.shape[1], 1)))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(LSTM(128, return_sequences=True)) # Lớp thứ 2
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(LSTM (256, return_sequences=True)) # Lớp thứ 3
    model.add(Dropout(0.3))
    model.add(LSTM(128)) # Lớp thứ 4
    model.add(Dense (128, activation='relu')) # Lớp thứ 5
    model.add(Dense(1)) # Đầu ra cho hồi quy
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
    return model
```

- Huấn luyện mô hình

```python
# 4. Huấn Luyện các mô hình
cnn_model = create_cnn_model()
rnn_model =  create_rnn_model()
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# Huấn Luyện CNN
cnn_history= cnn_model.fit(train_data, train_targets, epochs=50, batch_size=32, validation_split=0.1, callbacks=[early_stopping])
# Huấn Luyện RNN
rnn_history = rnn_model.fit(train_data, train_targets, epochs=50, batch_size=32, validation_split=0.1, callbacks=[early_stopping])
```

Python

```
:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument
 super().__init__(
:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
 super().__init__(**kwargs)
poch 1/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 489.9702 - mae: 20.0211 - val_loss: 443.1078 - val_mae: 20.0181
poch 2/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 118.9814 - mae: 8.3372 - val_loss: 448.4085 - val_mae: 20.1816
poch 3/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 60.2903 - mae: 5.8485 - val_loss: 447.2205 - val_mae: 20.1774
poch 4/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 56.8006 - mae: 5.5583 - val_loss: 429.6831 - val_mae: 19.7591
poch 5/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 47.3391 - mae: 4.8458 - val_loss: 415.4107 - val_mae: 19.4065
poch 6/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 40.1638 - mae: 4.5807 - val_loss: 396.7717 - val_mae: 18.9408
poch 7/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 40.7822 - mae: 4.7694 - val_loss: 383.8032 - val_mae: 18.5968
poch 8/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.5278 - mae: 4.3354 - val_loss: 361.2130 - val_mae: 18.0136
poch 9/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 31.5999 - mae: 3.9666 - val_loss: 344.2447 - val_mae: 17.5375
poch 10/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.9534 - mae: 4.4189 - val_loss: 330.8640 - val_mae: 17.1720
poch 11/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.5449 - mae: 4.0484 - val_loss: 313.3127 - val_mae: 16.6862
poch 12/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 34.2435 - mae: 4.2094 - val_loss: 298.5445 - val_mae: 16.2345
poch 13/50
..
poch 4/50
2/12 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 107.7668 - mae: 8.1556 - val_loss: 44.5440 - val_mae: 5.3986
```

- Đánh giá các mô hình

```
# 6. Đánh giá mô hình
cnn_mse = mean_squared_error(test_targets, cnn_predictions)
cnn_mae = mean_absolute_error(test_targets, cnn_predictions)
cnn_rmse = np.sqrt(cnn_mse) # Tính RMSE cho CNN
rnn_mse = mean_squared_error(test_targets, rnn_predictions)
rnn_mae = mean_absolute_error(test_targets, rnn_predictions)
rnn_rmse = np.sqrt(rnn_mse) # Tính RMSE cho RNN
```

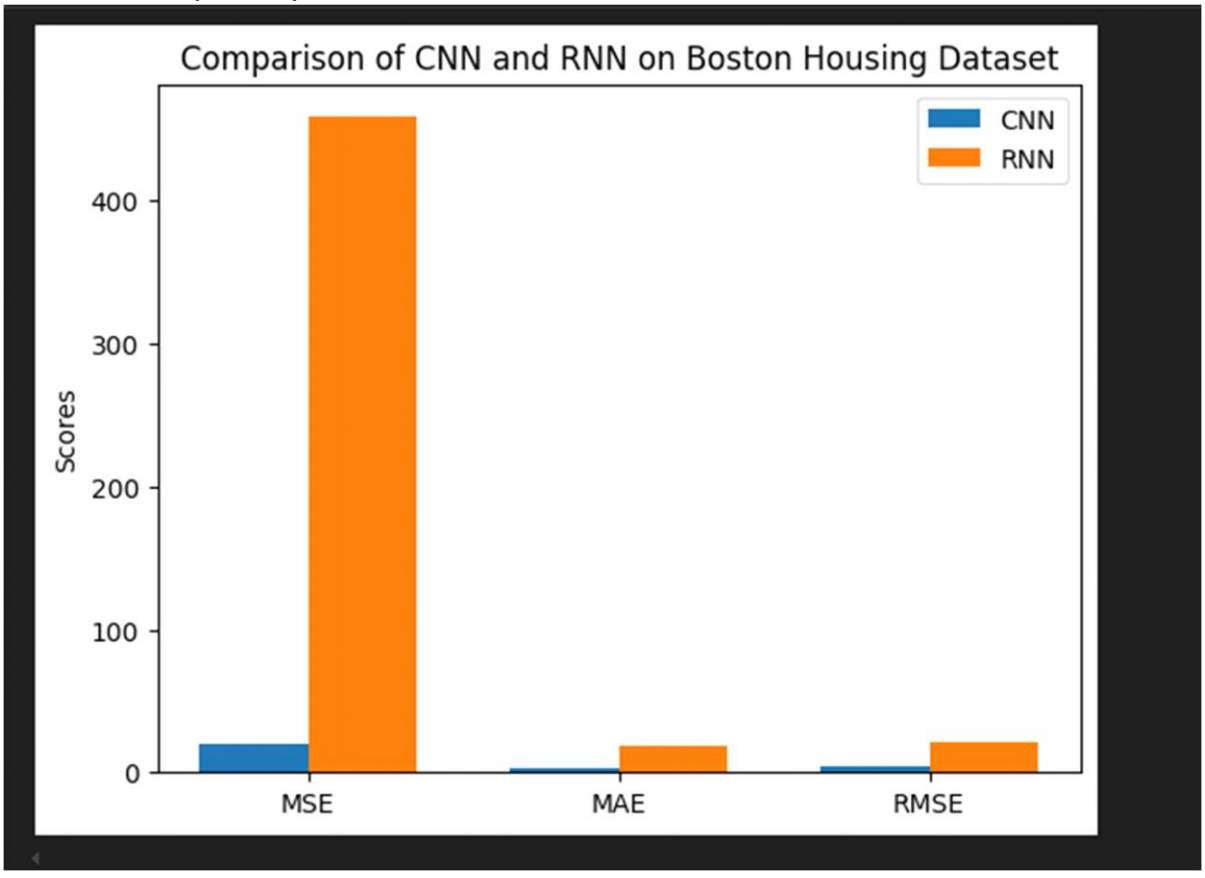
- Vẽ biểu đồ so sánh

```
# 7. Vẽ biểu đồ so sánh
labels = ['MSE', 'MAE', 'RMSE']
cnn_scores = [cnn_mse, cnn_mae, cnn_rmse]
rnn_scores = [rnn_mse, rnn_mae, rnn_rmse]
x = np.arange(len(labels)) # Vị trí nhãn
width = 0.35 # Độ rộng của các thanh
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, cnn_scores, width, label='CNN')
rects2 = ax.bar(x + width/2, rnn_scores, width, label='RNN')
# Thêm tiêu đề, nhãn và chú thích
ax.set_ylabel('Scores')
ax.set_title('Comparison of CNN and RNN on Boston Housing Dataset')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
# Hiển thị biểu đồ
plt.show()
```

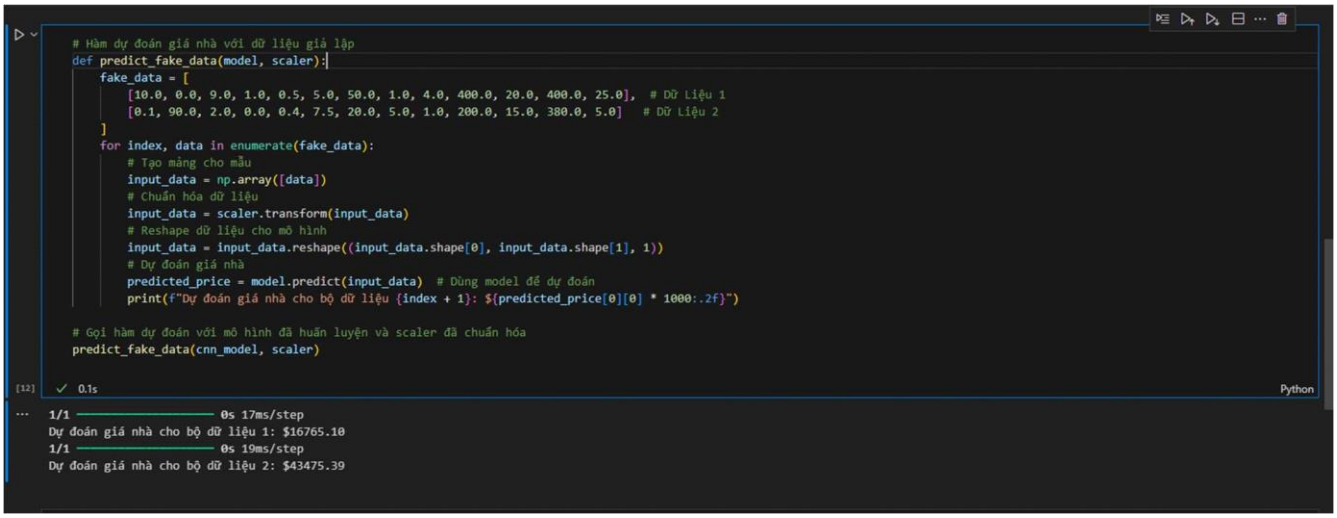
- Quá trình training

```
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(
c:\Users\Minnef\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
  super().__init__(**kwargs)
Epoch 1/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 489.9702 - mae: 20.0211 - val_loss: 443.1078 - val_mae: 20.0181
Epoch 2/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 118.9814 - mae: 8.3372 - val_loss: 448.4085 - val_mae: 20.1816
Epoch 3/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 60.2903 - mae: 5.8485 - val_loss: 447.2205 - val_mae: 20.1774
Epoch 4/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 56.8006 - mae: 5.5583 - val_loss: 429.6831 - val_mae: 19.7591
Epoch 5/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 47.3391 - mae: 4.8458 - val_loss: 415.4107 - val_mae: 19.4065
Epoch 6/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 40.1638 - mae: 4.5807 - val_loss: 396.7717 - val_mae: 18.9408
Epoch 7/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 40.7822 - mae: 4.7694 - val_loss: 383.8032 - val_mae: 18.5968
Epoch 8/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.5278 - mae: 4.3354 - val_loss: 361.2130 - val_mae: 18.0136
Epoch 9/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 31.5999 - mae: 3.9666 - val_loss: 344.2447 - val_mae: 17.5375
Epoch 10/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.9534 - mae: 4.4189 - val_loss: 330.8640 - val_mae: 17.1720
Epoch 11/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.5449 - mae: 4.0484 - val_loss: 313.3127 - val_mae: 16.6862
Epoch 12/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 34.2435 - mae: 4.2094 - val_loss: 298.5445 - val_mae: 16.2345
Epoch 13/50
...
Epoch 4/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 107.7668 - mae: 8.1556 - val_loss: 44.5440 - val_mae: 5.3986
Epoch 5/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 89.9894 - mae: 6.8503 - val_loss: 42.1543 - val_mae: 5.2747
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- Đồ thị vẽ được



- Dự đoán của mô hình



Giải thích

- Epoch
    - Epoch là một lần duyệt qua toàn bộ tập dữ liệu huấn luyện trong quá trình huấn luyện mô hình. Khi bạn đưa dữ liệu vào mô hình để học, thay vì cung cấp tất cả dữ liệu cùng một lúc, dữ liệu thường được chia thành các batch nhỏ để tăng hiệu suất tính toán. Một epoch kết thúc khi mô hình đã xử lý toàn bộ tập dữ liệu một lần qua các batch.
    - Ví dụ: Nếu bạn có 1,000 mẫu dữ liệu và sử dụng batch size là 100, thì sẽ cần 10 batch để hoàn thành một epoch.

- Batch
    - Batch là một tập hợp con nhỏ của dữ liệu được sử dụng để cập nhật trọng số của mô hình trong mỗi lần lặp (iteration). Thay vì cập nhật mô hình sau khi xử lý toàn bộ tập dữ liệu (gây tốn tài nguyên), mô hình sẽ cập nhật sau khi xử lý một batch. Điều này giúp quá trình huấn luyện nhanh hơn và giảm bớt bộ nhớ cần thiết.
    - Ví dụ: Nếu có 10,000 mẫu dữ liệu và batch size là 32, mô hình sẽ huấn luyện 32 mẫu mỗi lần lặp, và cần 312.5 lần lặp để hoàn thành một epoch.

- Training, Validation, Test Datasets
    - Training Dataset (Dữ liệu huấn luyện): Tập dữ liệu chính được sử dụng để huấn luyện mô hình. Mô hình học các đặc trưng và tối ưu trọng số từ tập này.
    - Validation Dataset (Dữ liệu xác nhận): Là một phần của tập dữ liệu không được sử dụng trong quá trình huấn luyện trực tiếp, nhưng dùng để đánh giá hiệu suất của mô hình trong quá trình huấn

luyện. Điều này giúp điều chỉnh các siêu tham số (như learning rate, batch size) và ngăn mô hình overfitting (quá khớp).

- Test Dataset (Dữ liệu kiểm tra): Tập dữ liệu không được sử dụng trong quá trình huấn luyện và xác nhận. Dữ liệu này được sử dụng để đánh giá cuối cùng mô hình sau khi nó đã được tối ưu hóa, cho phép đo lường khả năng tổng quát của mô hình trên dữ liệu chưa từng thấy.

- Training/Validation/Test Data + Validation là gì?

  - Training Data: Đây là dữ liệu được dùng để huấn luyện mô hình. Mô hình sẽ học từ các mẫu này để điều chỉnh các tham số.
  - Validation Data: Trong quá trình huấn luyện, validation data được dùng để kiểm tra hiệu suất của mô hình ở mỗi epoch hoặc batch. Đây là cách để phát hiện nếu mô hình bắt đầu overfitting hoặc underfitting.
    - Overfitting: Khi mô hình học quá kỹ vào dữ liệu huấn luyện đến mức nó không thể tổng quát hóa cho dữ liệu mới. Dấu hiệu overfitting thường là khi mô hình có hiệu suất tốt trên training data nhưng kém trên validation data.
    - Underfitting: Khi mô hình chưa đủ phức tạp để học tốt từ dữ liệu, dẫn đến kết quả kém trên cả training và validation data.

- Test là gì?

  - Test Data: Tập dữ liệu kiểm tra được sử dụng sau khi mô hình đã được huấn luyện và điều chỉnh. Nó không liên quan đến quá trình huấn luyện hay xác nhận. Dữ liệu này giúp đánh giá độ chính xác của mô hình trên dữ liệu mới mà mô hình chưa từng thấy. Kết quả trên test data là thước đo tốt nhất cho khả năng tổng quát hóa của mô hình trong thế giới thực.

Bài 4: Giải thích chi tiết về kiến trúc của CNN (Convolutional Neural Network) và RNN (Recurrent Neural Network):

- CNN (Convolutional Neural Network - Mạng Nơ-ron Tích Chập)
- Khái niệ

CNN là một loại mạng nơ-ron đặc biệt, được thiết kế để xử lý dữ liệu có cấu trúc dạng lưới, như hình ảnh. CNN sử dụng các lớp tích chập (convolutional layers) để trích xuất các đặc trưng quan trọng từ dữ liệu đầu vào.

- Kiến trúc

Kiến trúc CNN thường bao gồm các lớp chính sau:

  - Convolutional Layer (Lớp Tích Chập):

        - Đây là lớp quan trọng nhất của CNN. Lớp này thực hiện phép tích chập giữa bộ lọc (filter) với dữ liệu đầu vào, giúp trích xuất các đặc trưng (features) như cạnh, đường cong, màu sắc từ dữ liệu hình ảnh.

        - Mỗi bộ lọc trượt qua dữ liệu đầu vào và tạo ra một feature map (bản đồ đặc trưng). Bộ lọc có thể học được các đặc điểm như đường viền, góc, hoặc hình dạng.
    - Pooling Layer (Lớp Lấy Mẫu Xuống):
        - Sau mỗi lớp tích chập, lớp pooling giúp giảm kích thước của feature map, làm giảm số lượng tham số và tính toán, từ đó giúp mô hình đỡ phức tạp hơn. Lớp này thường sử dụng max pooling (lấy giá trị lớn nhất trong một vùng).
    - Flattening Layer (Lớp Làm Phẳng):
        - Sau khi qua nhiều lớp tích chập và pooling, dữ liệu thường được làm phẳng thành một vector để có thể đưa vào các lớp fully connected.
    - Fully Connected Layer (Lớp Kết Nối Đầy Đủ):
        - Đây là lớp cuối cùng của mô hình CNN, nơi các neuron được kết nối đầy đủ với nhau giống như mạng nơ-ron truyền thống (feedforward neural network). Lớp này sẽ thực hiện dự đoán cuối cùng dựa trên các đặc trưng đã được trích xuất.

- Ứng dụng

    - Nhận diện hình ảnh và video (image recognition)
    - Phân loại đối tượng (object classification)
    - Phát hiện đối tượng (object detection)
    - Nhận diện khuôn mặt, nhận diện ký tự (OCR)

- RNN (Recurrent Neural Network - Mạng Nơ-ron Hồi Quy)
- Khái niệm

RNN là một loại mạng nơ-ron đặc biệt dành cho dữ liệu có sự phụ thuộc thời gian hoặc tuần tự, như chuỗi thời gian, văn bản, âm thanh. Khác với mạng nơ-ron truyền thống, RNN có khả năng ghi nhớ các thông tin trước đó bằng cách chia sẻ trạng thái giữa các bước thời gian trong chuỗi.

- Kiến trúc

Kiến trúc của RNN khác biệt với CNN ở chỗ có sự kết nối giữa các bước thời gian. Các lớp chính của RNN bao gồm:

- Recurrent Layer (Lớp Hồi Quy):
    - Trong lớp này, mỗi neuron không chỉ nhận đầu vào từ bước thời gian hiện tại mà còn từ trạng thái của neuron ở bước thời gian trước đó. Điều này giúp RNN có khả năng ghi nhớ các thông tin trước đó để dự đoán bước tiếp theo.
- Hidden State (Trạng Thái Ẩn):
    - Trạng thái ẩn là nơi lưu trữ các thông tin đã học từ các bước trước đó. Tại mỗi bước thời gian, thông tin đầu vào và trạng thái ẩn được cập nhật dựa trên một hàm hồi quy.
- Vanishing Gradient Problem:

    - Một vấn đề lớn của RNN là vanishing gradient (độ dốc biến mất), làm giảm khả năng học các phụ thuộc dài hạn. Để giải quyết vấn đề này, các biến thể của RNN như LSTM (Long Short-Term Memory) và GRU (Gated Recurrent Unit) đã ra đời.

- Các biến thể của RNN

    - LSTM (Long Short-Term Memory):
        - LSTM khắc phục vấn đề vanishing gradient bằng cách sử dụng các cổng (gates) để kiểm soát luồng thông tin qua các trạng thái ẩn. Điều này giúp mô hình học được các phụ thuộc dài hạn trong chuỗi dữ liệu.
    - GRU (Gated Recurrent Unit):
        - GRU cũng tương tự như LSTM nhưng có cấu trúc đơn giản hơn, giúp giảm thiểu số lượng tính toán và tăng hiệu suất cho các mô hình RNN.

- Ứng dụng

    - Xử lý ngôn ngữ tự nhiên (NLP): dịch máy, phân loại văn bản, nhận dạng giọng nói
    - Dự đoán chuỗi thời gian: dự báo giá cổ phiếu, phân tích dữ liệu thời tiết
    - Nhận diện giọng nói, âm thanh