Họ và tên: Nguyễn Việt Quang

MSV: B22DCCN650

Lớp: D22CNPM01

Câu 1

1. Lịch sử phát triển

CNN

- Ý tưởng mạng tích chập được hình thành trong lĩnh vực thị giác máy tính với "lớp receptive field" và các bộ lọc (kernel) từ những năm 1980. aiex.ai+2VAST Journals+2
- Một trong các mô hình ban đầu nổi bật: LeNet- 5 của Yann LeCun - dùng để nhận diện chữ viết tay (ZIP code) đầu thập niên 1990. Studocu+1
- Đến năm 2012, với mô hình AlexNet thắng cuộc thi ImageNet làm bùng nổ việc dùng CNN sâu trong computer vision. Reddit

RNN

- Mạng hồi tiếp (Recurrent Neural Network) được phát triển để xử lý dữ liệu tuần tự (chuỗi thời gian, văn bản, âm thanh). Ví dụ khởi đầu từ mạng của John Hopfield năm 1982. Wikipedia+1
- Tuy nhiên RNN truyền thống gặp vấn đề về **gradient biến mất/loãng** (vanishing/exploding gradient) khi học các chuỗi dài.

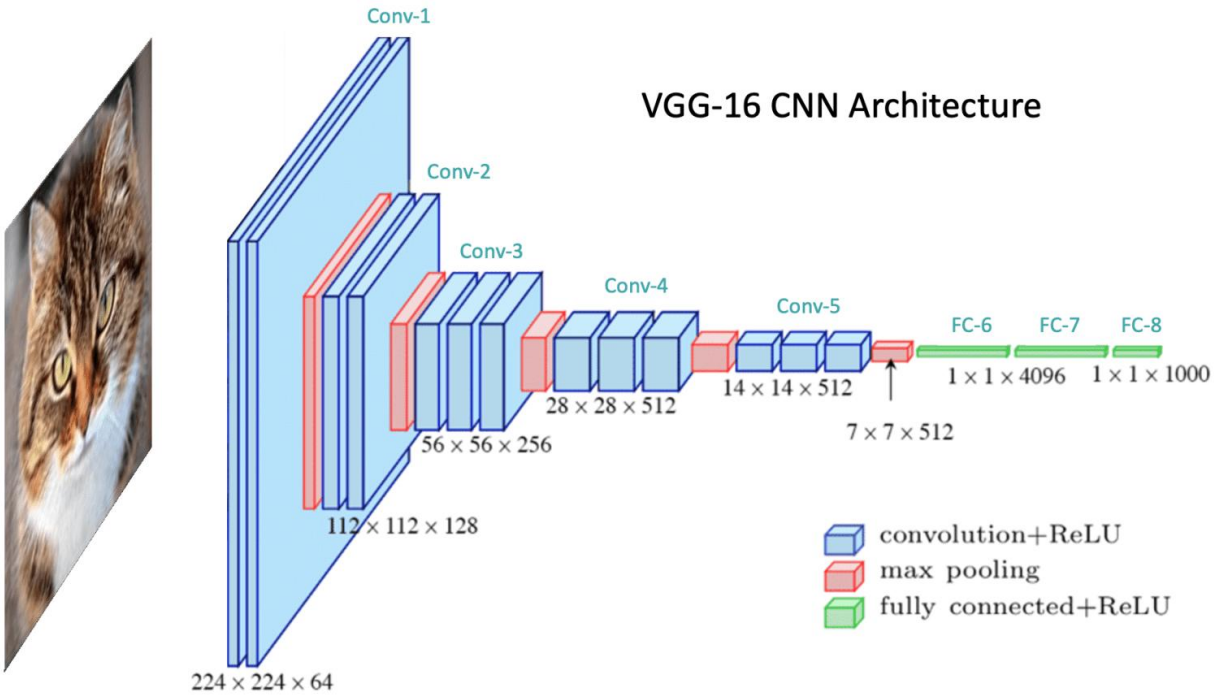
LSTM

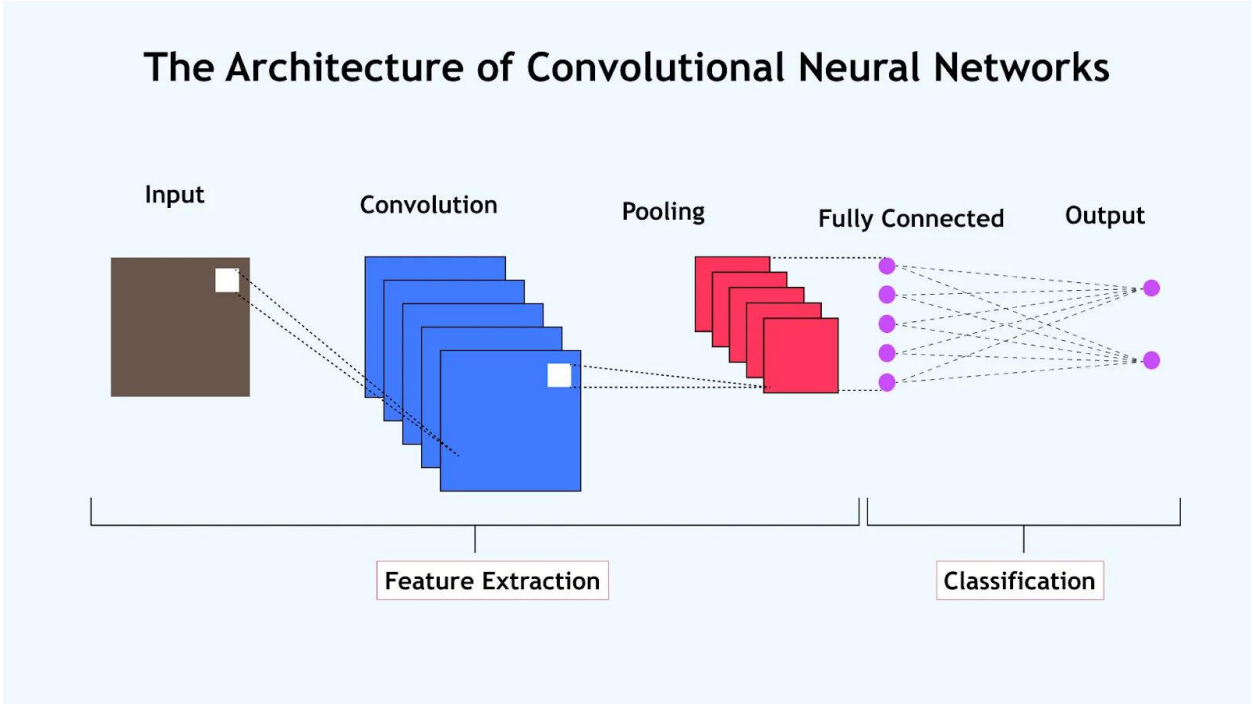
- Để khắc phục nhược điểm của RNN trên chuỗi dài, vào năm 1997, Sepp Hochreiter & Jürgen Schmidhuber đề xuất LSTM – mạng có "cell trạng thái dài" và các cổng (gates) để ghi nhớ/xóa thông tin. Wikipedia+2Science Publications+2
- Từ đó LSTM được ứng dụng rộng rãi trong xử lý ngôn ngữ, nhận diện giọng nói, chuỗi thời gian. PMC+1

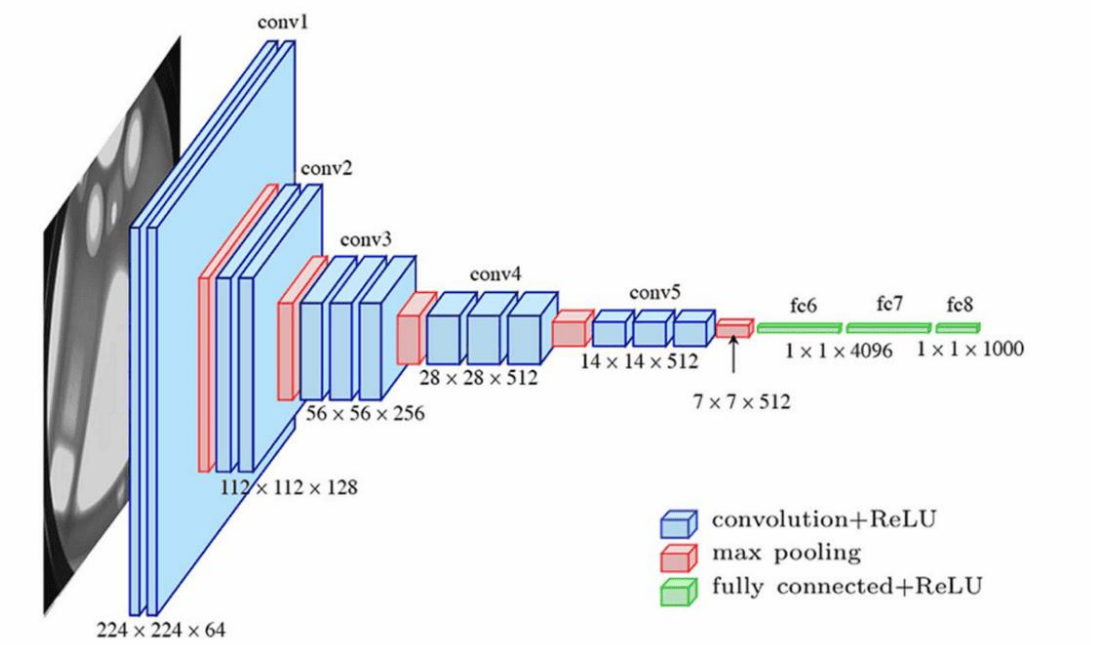
Bạn sẽ dùng phần này làm "1 trang" (giới thiệu lịch sử, các mốc lớn, ai/nhóm nào làm, vì sao cần ra đời mô hình mới).

2. Kiến trúc của các mô hình

CNN


VGG-16 CNN Architecture
Conv-1
Conv-2
Conv-3
Conv-4
Conv-5
FC-6
FC-7
FC-8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU


The Architecture of Convolutional Neural Networks
Input
Convolution
Pooling
Fully Connected
Output
Feature Extraction
Classification

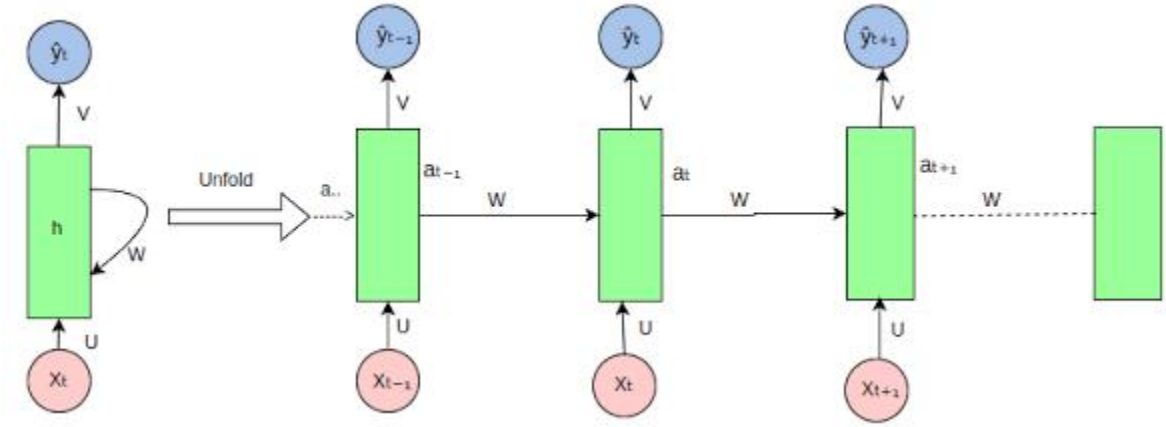


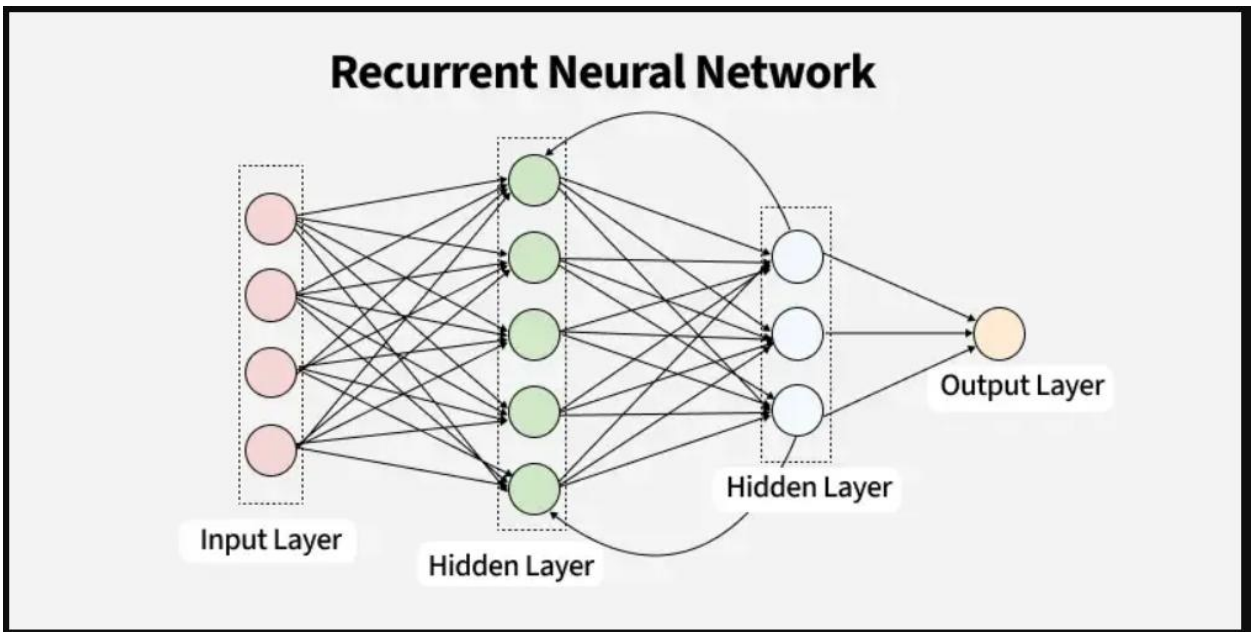
6

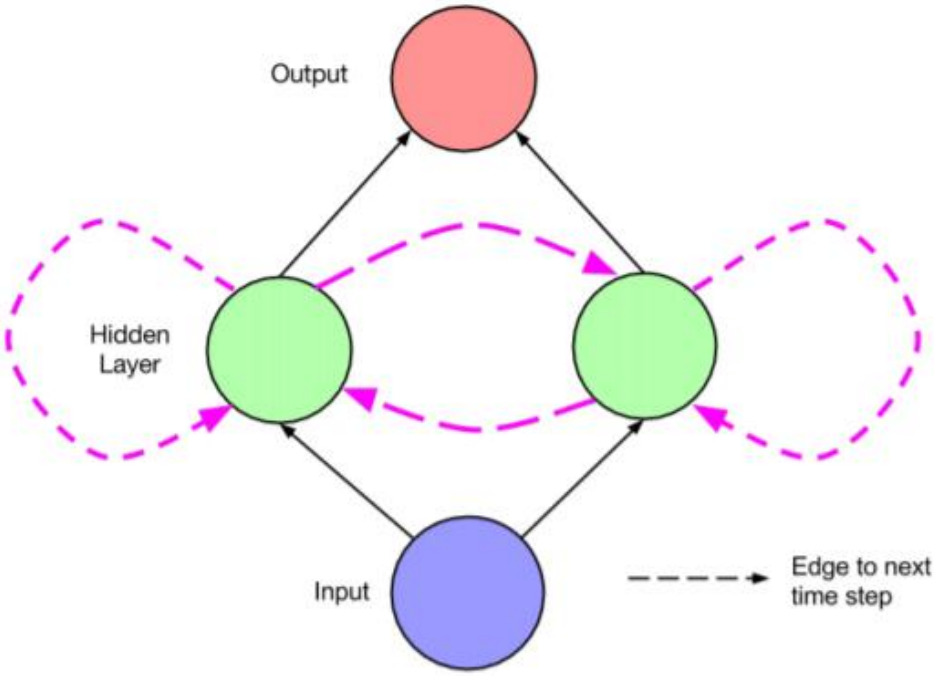
- Nhìn chung, CNN gồm các khối: **Lớp tích chập (Convolutional layer) → Lớp gộp/giảm kích thước (Pooling layer: MaxPool / AvgPool)** → (có thể lặp lại nhiều lần) → **Lớp nối đầy (Fully Connected / Dense layer)** → lớp đầu ra (softmax hoặc sigmoid). MDPI+1
- Ví dụ code minh hoạ (Keras/TensorFlow):
- from tensorflow.keras import layers, models
-
- model = models.Sequential()
- model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(height, width, channels)))
- model.add(layers.MaxPooling2D((2,2)))
- model.add(layers.Conv2D(64, (3,3), activation='relu'))
- model.add(layers.MaxPooling2D((2,2)))
- model.add(layers.Conv2D(64, (3,3), activation='relu'))
- # → flatten + dense
- model.add(layers.Flatten())

- model.add(layers.Dense(64, activation='relu'))
- model.add(layers.Dense(num_classes, activation='softmax'))
- model.summary()
- Trong code trên:
    - Mũi tên ⚫: giữa các lớp Conv2D và MaxPooling2D thể hiện "giảm spatial size nhưng tăng depth (số filter)"
    - Mũi tên ⚫: từ flatten → dense là "chuyển trích xuất đặc trưng sang phân loại"

RNN







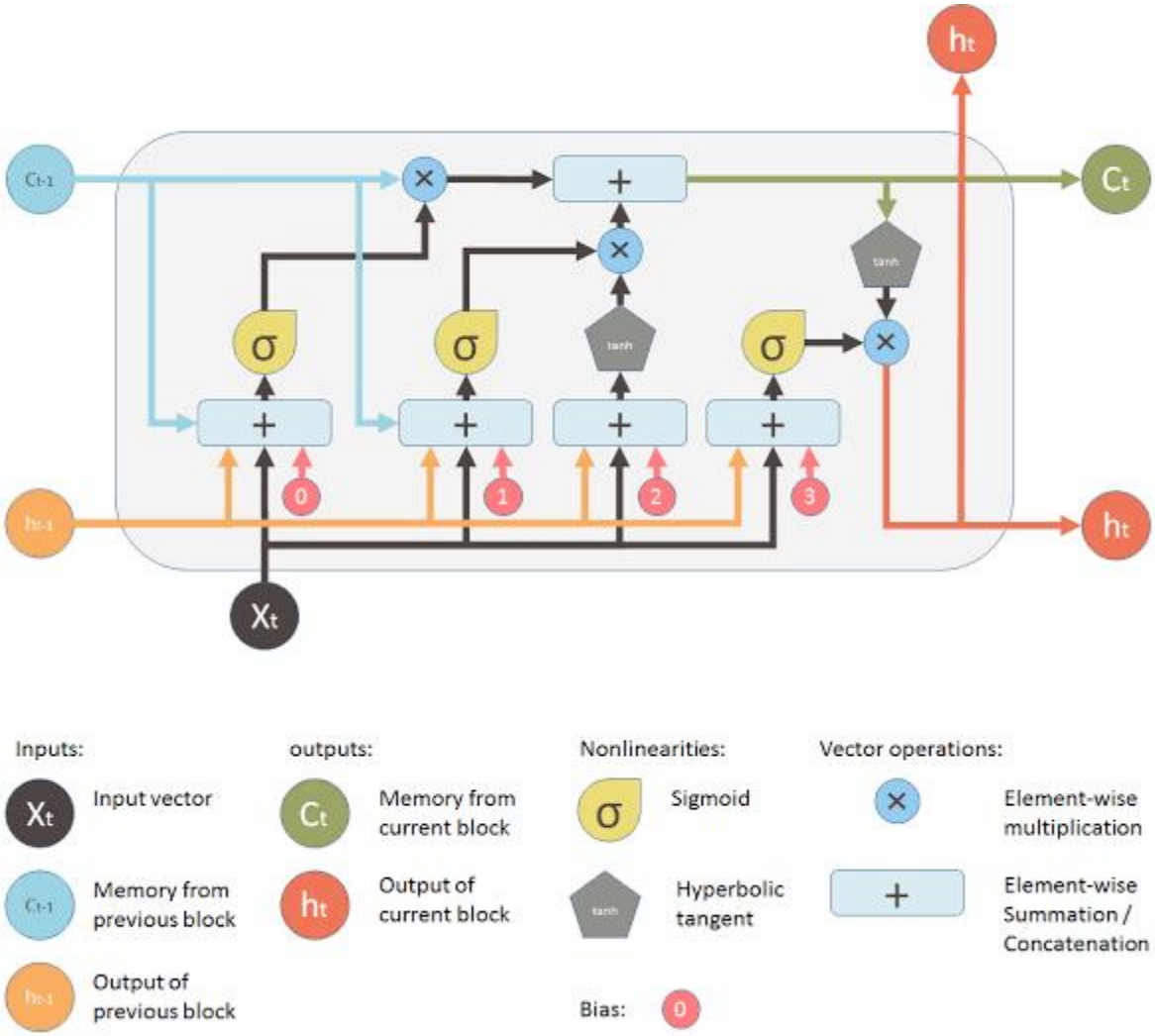
6

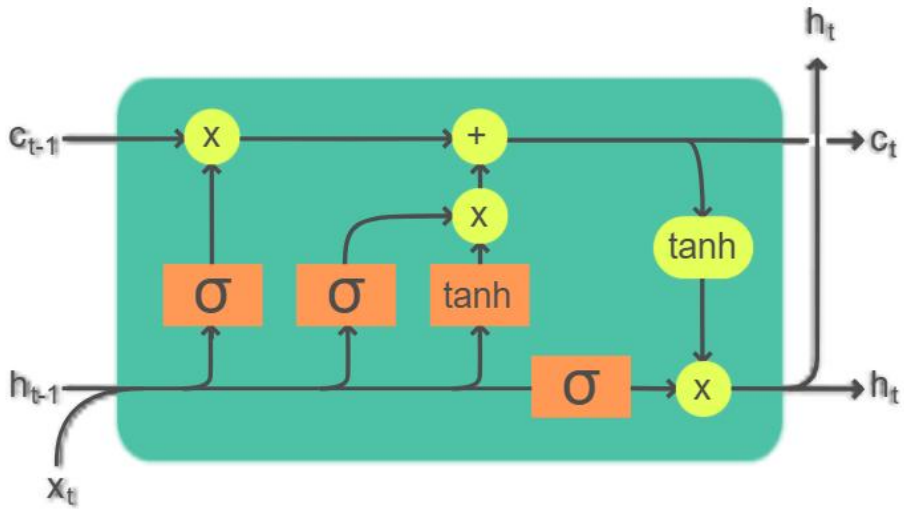
- Kiến trúc RNN: tại mỗi bước thời gian t, mạng nhận đầu vào $x_t$và trạng thái ẩn h_{t-1}, rồi xuất $h_t$và/hoặc y_t. Có kết nối hồi tiếp (loop) trong cùng một lớp. oksim.ua
- Ví dụ code minh hoạ (Keras):
- from tensorflow.keras import layers, models

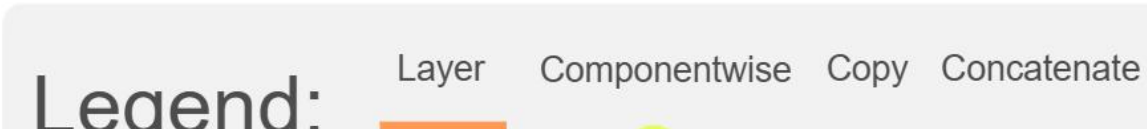
- model = models.Sequential()
- model.add(layers.SimpleRNN(50, activation='tanh', input_shape=(timesteps, features)))
- model.add(layers.Dense(1, activation='sigmoid'))
- model.summary()
- Trong code:
    - Mũi tên ⚫: từ input_shape → SimpleRNN là “nhận chuỗi dữ liệu có timesteps”

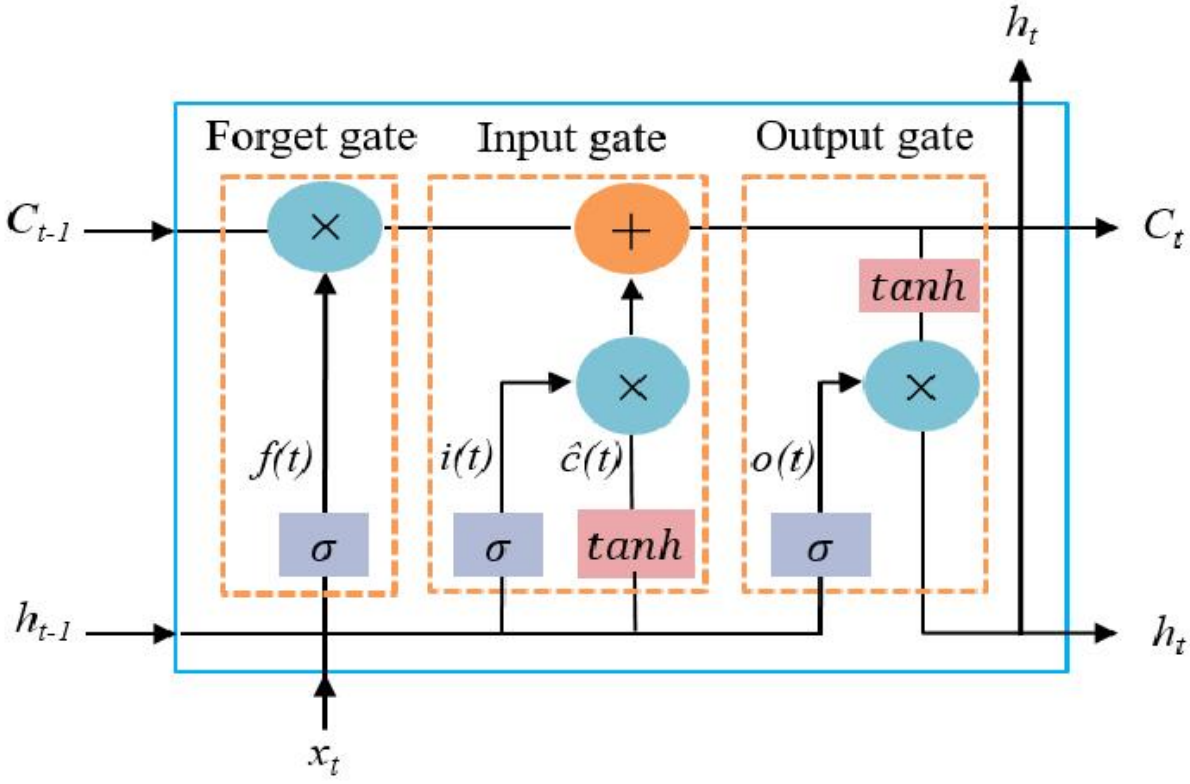
- Mũi tên ⬤: từ SimpleRNN → Dense là "tóm gọn toàn bộ thông tin chuỗi thành output phân loại"

LSTM




h_t
c_{t-1}
x
+
c_t
x
tanh
σ
σ
tanh
σ
x
h_{t-1}
h_t
x_t


Legend:
Layer
Componentwise
Copy
Concatenate


h_t
Forget gate
Input gate
Output gate
C_{t-1}
×
+
C_t
tanh
×
×
$f(t)$
$i(t)$
$\hat{c}(t)$
$o(t)$
σ
σ
tanh
σ
h_{t-1}
h_t
x_t

- Kiến trúc LSTM: mỗi "cell" có một trạng thái cell c_t, một trạng thái ẩn h_t, và ba (hoặc nhiều) cổng: **forget gate**, **input gate**, **output gate**. Cấu trúc này giúp mạng lưu thông tin lâu hơn và tránh vấn đề gradient biến mất. Wikipedia+1
- Ví dụ code minh hoạ (Keras):
- from tensorflow.keras import layers, models

- model = models.Sequential()
- model.add(layers.LSTM(100, input_shape=(timesteps, features)))
- model.add(layers.Dense(num_classes, activation='softmax'))
- model.summary()
- Trong code:
    - Mũi tên : từ input_shape → LSTM là "chuỗi dữ liệu được đưa vào tế bào LSTM"
    - Mũi tên : từ LSTM → Dense là "tóm gọn thông tin chuỗi thành đầu ra phân loại"

3. Code kiến trúc tương ứng

Dưới đây là **một đoạn code mẫu** cho mỗi mô hình (gắn thêm chú thích mũi tên màu đỏ) — bạn có thể copy và chỉnh cho phù hợp với bài báo cáo (với indent, font code…).

CNN code mẫu

from tensorflow.keras import layers, models

model = models.Sequential()

Lớp tích chập đầu tiên: học filter từ ảnh đầu vào

model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(64,64,3)))

Lớp pooling: giảm kích thước không gian, giữ đặc trưng mạnh

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

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

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

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

model.add(layers.Flatten())  # 🔘 chuyển từ không gian 2D sang vector 1D
model.add(layers.Dense(64, activation='relu'))  # 🔘 lớp fully connected
model.add(layers.Dense(10, activation='softmax'))  # 🔘 lớp đầu ra phân loại 10 lớp

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

RNN code mẫu

```
from tensorflow.keras import layers, models

model = models.Sequential()
# LSTM có thể dùng thay SimpleRNN – nhưng ở đây dùng RNN đơn giản cho minh hoạ
model.add(layers.SimpleRNN(50, activation='tanh', input_shape=(100, 1)))
# 100 timesteps, mỗi timestep có 1 đặc trưng
model.add(layers.Dense(1, activation='sigmoid'))  # đầu ra nhị phân

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

LSTM code mẫu

```
from tensorflow.keras import layers, models

model = models.Sequential()
# LSTM học từ chuỗi dữ liệu với khả năng nhớ lâu hơn
model.add(layers.LSTM(100, input_shape=(100, 1)))
# 100 timesteps, mỗi timestep 1 đặc trưng
```

```
model.add(layers.Dense(1, activation='sigmoid'))  # đầu ra nhị phân


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

4. Chỉ ra sự khác biệt của 3 mô hình qua Hình ảnh, Code

Sự khác biệt chính

- **Dữ liệu thích hợp**:
    - CNN chủ yếu xử lý dữ liệu có không gian (spatial) như ảnh, video (2D hoặc 3D).
    - RNN / LSTM chủ yếu xử lý dữ liệu tuần tự (temporal) như chuỗi thời gian, văn bản, âm thanh.
- **Kiến trúc kết nối**:
    - CNN: kết nối từ trái → phải, không có vòng quay lại (no recurrence).
    - RNN: có "vòng lặp" qua thời gian (hidden state $h_{t-1} \rightarrow h_t$).
    - LSTM: cũng giống RNN có vòng lặp, nhưng cell có trạng thái dài, và các cổng điều khiển việc lưu/xóa thông tin.
- **Khả năng nhớ lâu**:
    - RNN truyền thống khó nhớ các mối liên hệ ở khoảng thời gian dài vì vanishing gradient.
    - LSTM khắc phục bằng thiết kế cổng và cell state.
- **Ứng dụng**:
    - CNN: nhận dạng hình ảnh, phân loại ảnh, detection, segmentation.
    - RNN/LSTM: dịch máy, phân tích cảm xúc, dự báo chuỗi thời gian, nhận diện giọng nói.
- **Code/thiết kế**:
    - Trong code, CNN: dùng Conv2D/Pooling → Flatten → Dense.
    - RNN: dùng SimpleRNN (hoặc RNN) với input_shape=(timesteps, features).

    - LSTM: dùng LSTM layer tương tự RNN nhưng "units" thường lớn hơn, thường đặt return_sequences=True nếu deep stack, hoặc return_state=True.
- **Hình ảnh minh họa**: bạn có thể gắn các mũi tên màu đỏ () chỉ vào chỗ khác biệt, ví dụ: trong hình CNN chỉ ra lớp pooling, trong hình RNN chỉ ra mối liên kết thời gian, trong hình LSTM chỉ ra các cổng forget/input/output.

Ví dụ cụ thể qua hình & code

- Hình CNN: mũi tên đỏ chỉ vào "Pooling layer" → tiết kiệm tham số, giữ đặc trưng.
- Hình RNN: mũi tên đỏ chỉ vào "loop" từ h_{t-1} → h_t (hidden state).
- Hình LSTM: mũi tên đỏ chỉ vào "forget gate" và "cell state" chuyển tiếp (c_{t-1} → c_t) để nhấn mạnh khác biệt với RNN.
- Code:
    - Trong CNN code: thêm comment và mũi tên đỏ (qua chú thích) như "Flatten chuyển sang vector 1D"
    - Trong RNN code: "input_shape=(timesteps, features) cho chuỗi"
    - Trong LSTM code: "LSTM(100) – nhớ lâu hơn RNN".

Câu 2

Pha 1: Chuẩn bị dữ liệu (Data Preparation)

Trong pha này, dữ liệu thời tiết Việt Nam được tải và xử lý để phù hợp cho việc huấn luyện mô hình học sâu. Các bước bao gồm:

- Đọc dữ liệu từ tệp weather_2.csv.
- Chọn các cột cần thiết như Date, Location, Temperature, Humidity, Rainfall, v.v.
- Xử lý giá trị thiếu và chuẩn hóa dữ liệu bằng MinMaxScaler.
- Chia dữ liệu thành tập huấn luyện và tập kiểm thử (train/test split) theo từng thành phố: Hà Nội, Đà Nẵng, Thành phố Hồ Chí Minh.
- Tạo các chuỗi thời gian (sequence) làm đầu vào cho mô hình RNN/LSTM, ví dụ: sử dụng 7 ngày gần nhất để dự đoán ngày tiếp theo.

```python
# ================================================
# PHASE 1: DATA PREPARATION & EXPLORATION
# ================================================

def load_and_inspect(path):
    """Load dataset and print structure for exploration"""
    if not os.path.exists(path):
        raise FileNotFoundError(f"File not found: {path}")
    df = pd.read_csv(path)
    print("\n-- Dataset Info --")
    print(df.info())
    print("\n-- Missing values --")
    print(df.isna().sum())
    print("\n-- Head --")
    print(df.head())
    return df

def normalize_city_name(s):
    s = str(s).strip().lower()
    if s in [c.lower() for c in CITIES]:
        return 'Hanoi'
    if s in [c.lower() for c in DANANG_VARIANTS]:
        return 'Danang'
    if s in [c.lower() for c in HCM_VARIANTS]:
        return 'HCMC'
    return s
```

```python
def prepare_city_df(df, city_label='Hanoi'):
    """Filter dataset by city and convert date"""
    date_col = [c for c in df.columns if 'date' in c.lower()][0]
    city_col = [c for c in df.columns if 'city' in c.lower() or 'location' in c.lower()][0]
    df[date_col] = pd.to_datetime(df[date_col])
    df['city_norm'] = df[city_col].apply(normalize_city_name)
    city_df = df[df['city_norm'] == city_label].sort_values(date_col)
    return city_df.reset_index(drop=True), date_col

def find_numeric_feature_candidates(df):
    exclude = [c for c in df.columns if 'date' in c.lower() or 'city' in c.lower()]
    return [c for c in df.select_dtypes(include=[np.number]).columns if c not in exclude]
```

Pha 2: Thiết kế và huấn luyện mô hình (Model Design & Training)

Trong pha này, hai mô hình học sâu được xây dựng để dự đoán nhiệt độ:

- **Mô hình RNN (Recurrent Neural Network)**: sử dụng các tầng SimpleRNN để xử lý chuỗi thời gian.
- **Mô hình LSTM (Long Short-Term Memory)**: cải tiến so với RNN, có khả năng ghi nhớ dài hạn giúp tăng độ chính xác trong dự báo thời tiết.

Các bước thực hiện:

- Khởi tạo kiến trúc mô hình với các tầng RNN hoặc LSTM, Dense đầu ra để dự đoán nhiệt độ.
- Cấu hình hàm mất mát (loss='mse') và bộ tối ưu hóa (optimizer='adam').
- Huấn luyện mô hình với dữ liệu huấn luyện, lưu lại lịch sử huấn luyện (history) để trực quan hóa quá trình học.

```
# ================================================
# PHASE 2: MODELING (RNN & LSTM)
# ================================================

def create_sequences(values, seq_len):
    """Convert time-series data into sequences for RNN/LSTM"""
    X, y = [], []
    for i in range(len(values) - seq_len):
        X.append(values[i:i+seq_len])
        y.append(values[i+seq_len, 0])  # predict target of next day
    return np.array(X), np.array(y)

def build_rnn_model(input_shape, units=32, lr=0.001):
    """Simple RNN model"""
    model = Sequential([
        SimpleRNN(units, input_shape=input_shape),
        Dense(16, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=lr), loss='mse', metrics=['mae'])
    return model
```

```python
def build_lstm_model(input_shape, units=50, lr=0.001):
    """LSTM model"""
    model = Sequential([
        LSTM(units, input_shape=input_shape),
        Dense(16, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=lr), loss='mse', metrics=['mae'])
    return model

def run_pipeline_for_city(df_city, date_col, features, target_col, seq_len=14, test_ratio=0.15, val_ratio=0.15):
    """Train & evaluate RNN and LSTM for one city"""
    data = df_city[features].dropna()
    scaler = MinMaxScaler()
    scaled = scaler.fit_transform(data)

    X, y = create_sequences(scaled, seq_len)
    n = len(X)
    test_size, val_size = int(n*test_ratio), int(n*val_ratio)
    X_train, X_val, X_test = X[:-val_size-test_size], X[-val_size-test_size:-test_size], X[-test_size:]
    y_train, y_val, y_test = y[:-val_size-test_size], y[-val_size-test_size:-test_size], y[-test_size:]

    # --- RNN Training ---
    rnn = build_rnn_model((X_train.shape[1], X_train.shape[2]))
    es = EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)
    hist_rnn = rnn.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=32, verbose=0, callbacks=[es])

    # --- LSTM Training ---
    lstm = build_lstm_model((X_train.shape[1], X_train.shape[2]))
    hist_lstm = lstm.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=32, verbose=0, callbacks=[es])
```

```python
    # --- Prediction ---
    def inv_transform(pred):
        dummy = np.zeros((len(pred), scaled.shape[1]))
        dummy[:,0] = pred
        return scaler.inverse_transform(dummy)[:,0]

    pred_rnn = inv_transform(rnn.predict(X_test).flatten())
    pred_lstm = inv_transform(lstm.predict(X_test).flatten())
    y_true = inv_transform(y_test)

    # --- Metrics ---
    metrics = {}
    for name, pred in [('RNN', pred_rnn), ('LSTM', pred_lstm)]:
        mae = mean_absolute_error(y_true, pred)
        mse = mean_squared_error(y_true, pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_true, pred)
        metrics[name] = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R2': r2}

    return hist_rnn, hist_lstm, y_true, pred_rnn, pred_lstm, metrics
```

Pha 3: Đánh giá và triển khai mô hình (Evaluation & Deployment)

Sau khi huấn luyện, mô hình được đánh giá và so sánh dựa trên các độ đo:

- **MAE (Mean Absolute Error)**: sai số trung bình tuyệt đối.
- **MSE (Mean Squared Error)**: sai số bình phương trung bình.
- **RMSE (Root Mean Squared Error)**: căn bậc hai của MSE.
- **R^2 (Hệ số xác định)**: đo mức độ phù hợp của mô hình.

Biểu đồ được sử dụng để trực quan hóa:

- Quá trình huấn luyện (training loss vs validation loss).
- So sánh giá trị dự đoán và giá trị thực tế của từng thành phố.
- Biểu đồ cột thể hiện sự khác biệt giữa RNN và LSTM theo các độ đo.

```
# ================================================
# PHASE 3: EVALUATION & VISUALIZATION
# ================================================

def plot_training(history, title):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.title(title)
    plt.xlabel('Epoch')
    plt.ylabel('MSE')
    plt.legend()
    plt.show()

def plot_predictions(y_true, pred_rnn, pred_lstm, city):
    plt.figure(figsize=(10,5))
    plt.plot(y_true, label='Actual')
    plt.plot(pred_rnn, label='RNN')
    plt.plot(pred_lstm, label='LSTM')
    plt.title(f'Predictions vs Actual ({city})')
    plt.legend()
    plt.show()
```

```python
def plot_predictions(y_true, pred_rnn, pred_lstm, city):
    plt.figure(figsize=(10,5))
    plt.plot(y_true, label='Actual')
    plt.plot(pred_rnn, label='RNN')
    plt.plot(pred_lstm, label='LSTM')
    plt.title(f'Predictions vs Actual ({city})')
    plt.legend()
    plt.show()

def plot_comparison(metrics, city):
    models = list(metrics.keys())
    maes = [metrics[m]['MAE'] for m in models]
    mses = [metrics[m]['MSE'] for m in models]
    plt.bar(models, maes, label='MAE')
    plt.bar(models, mses, alpha=0.7, label='MSE')
    plt.title(f'Model Comparison ({city})')
    plt.legend()
    plt.show()
```

MAIN PIPELINE

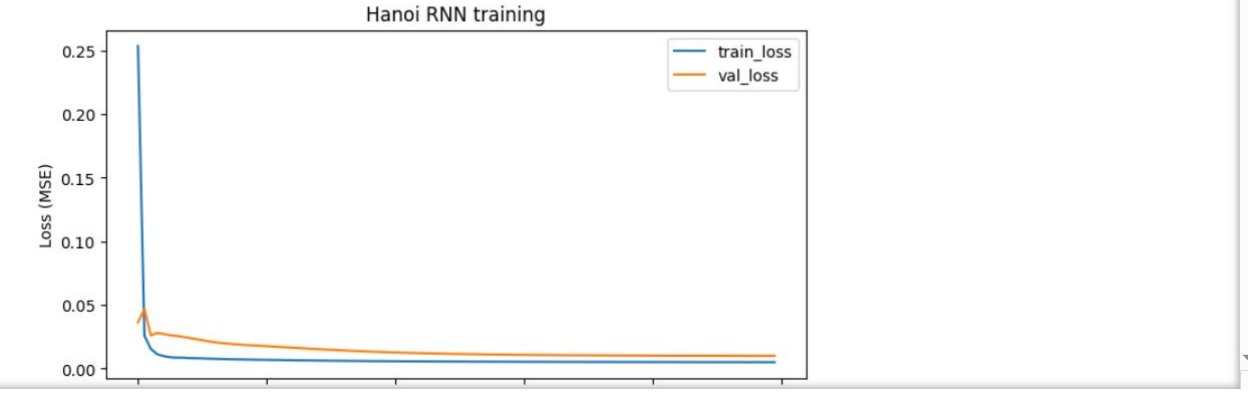
```python
def main():
    df = load_and_inspect(DATA_PATH)
    cities = {'Hanoi': None, 'Danang': None, 'HCMC': None}

    # --- Prepare data for each city ---
    for city in cities.keys():
        try:
            df_city, date_col = prepare_city_df(df, city)
            if df_city.empty:
                continue
            features = [TARGET_COLUMN] + find_numeric_feature_candidates(df_city)[:3]
            print(f'City: {city}, features: {features}')

            hist_rnn, hist_lstm, y_true, pred_rnn, pred_lstm, metrics = run_pipeline_for_city(df_city, date_col, features, TARGET_COLUMN)

            print(f'Metrics for {city}:', metrics)

            # --- Plotting ---
            plot_training(hist_rnn, f'{city} RNN Training')
            plot_training(hist_lstm, f'{city} LSTM Training')
            plot_predictions(y_true, pred_rnn, pred_lstm, city)
            plot_comparison(metrics, city)
        except Exception as e:
            print(f'Skipping {city}: {e}')

if __name__ == '__main__':
    main()
```

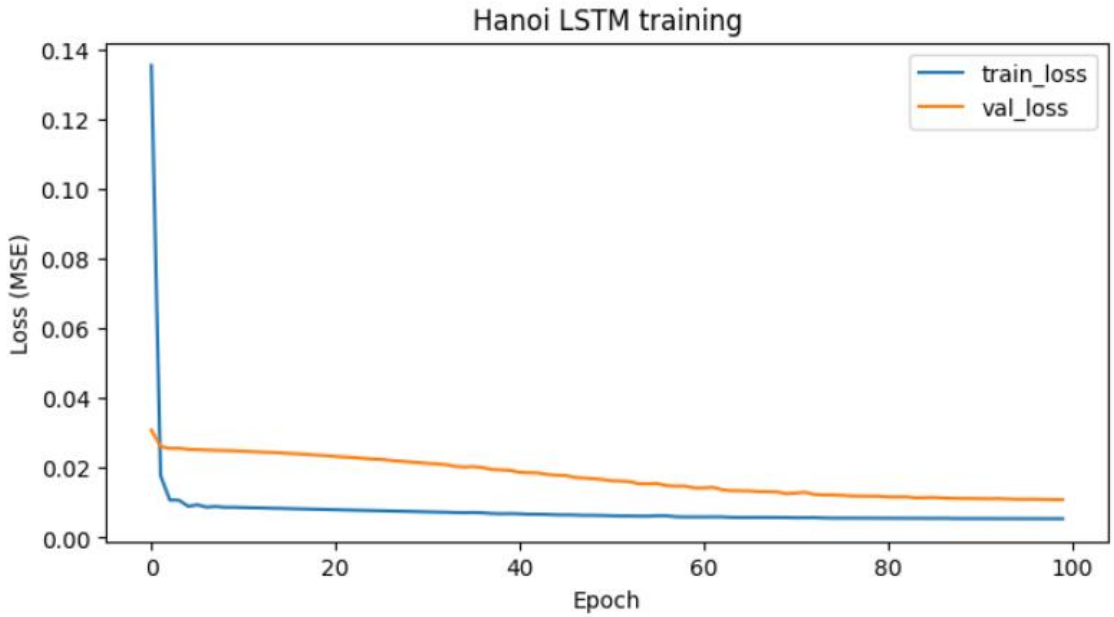
Kết luận

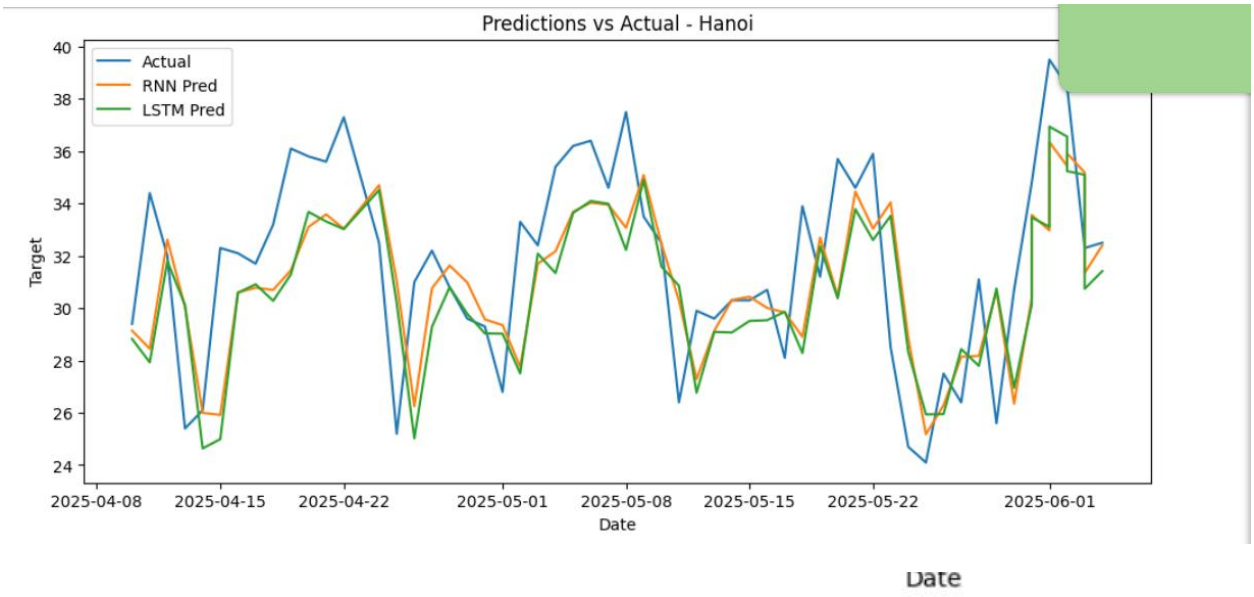
Việc áp dụng học sâu trong dự báo thời tiết cho thấy hiệu quả rõ rệt, đặc biệt khi sử dụng mô hình LSTM. Với dữ liệu đủ lớn và tiền xử lý tốt, mô hình có thể được triển khai cho dự báo thời tiết tự động tại các khu vực khác nhau.

Output

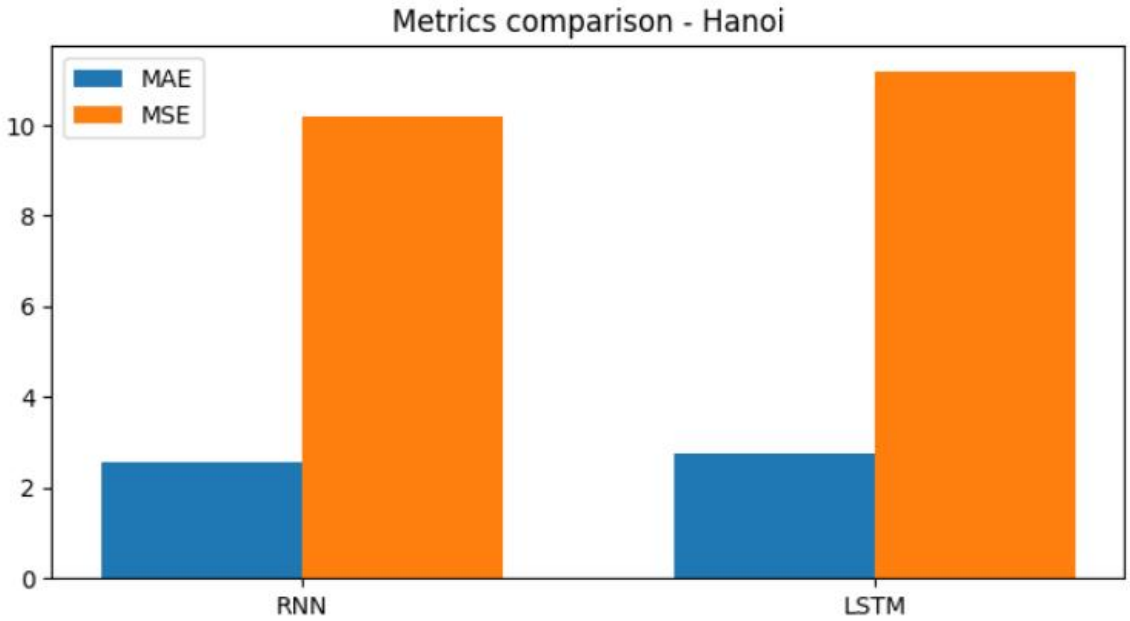
```
Metrics for Hanoi:
{'RNN': {'MAE': 2.5767081408177375, 'MSE': 10.189694359643175, 'RMSE': 3.1921300662164716, 'R2': 0.31995379359689}, 'LSTM': {'MAE': 2.75222092280953
4, 'MSE': 11.182636709900004, 'RMSE': 3.344044962302392, 'R2': 0.2536861849094798}}
```





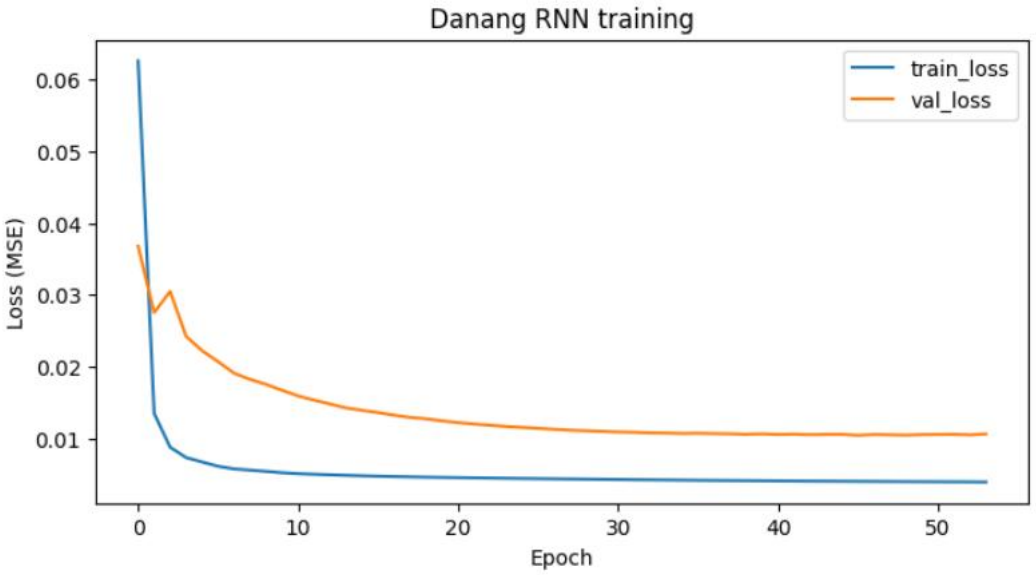


Date

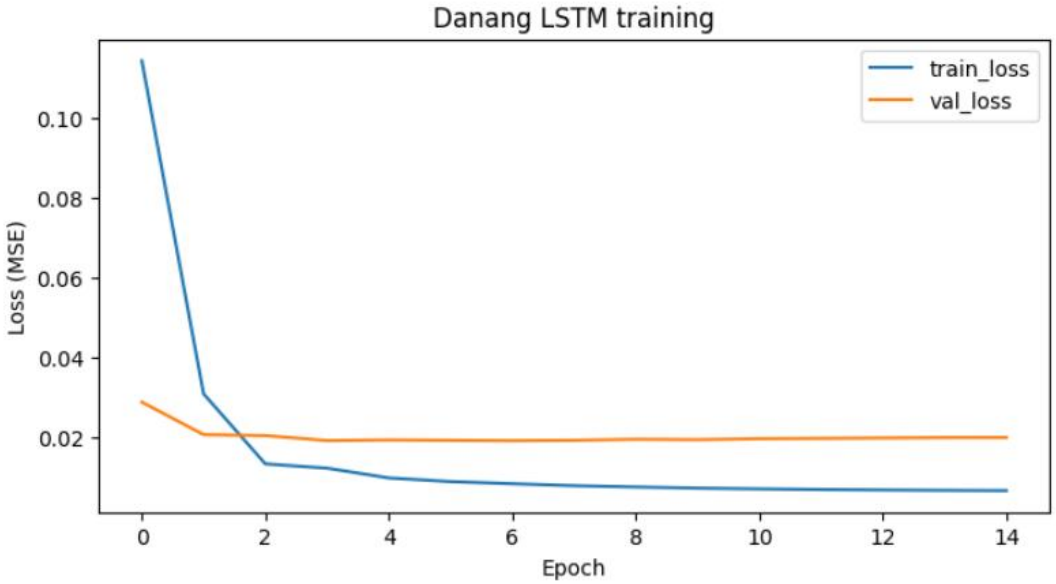


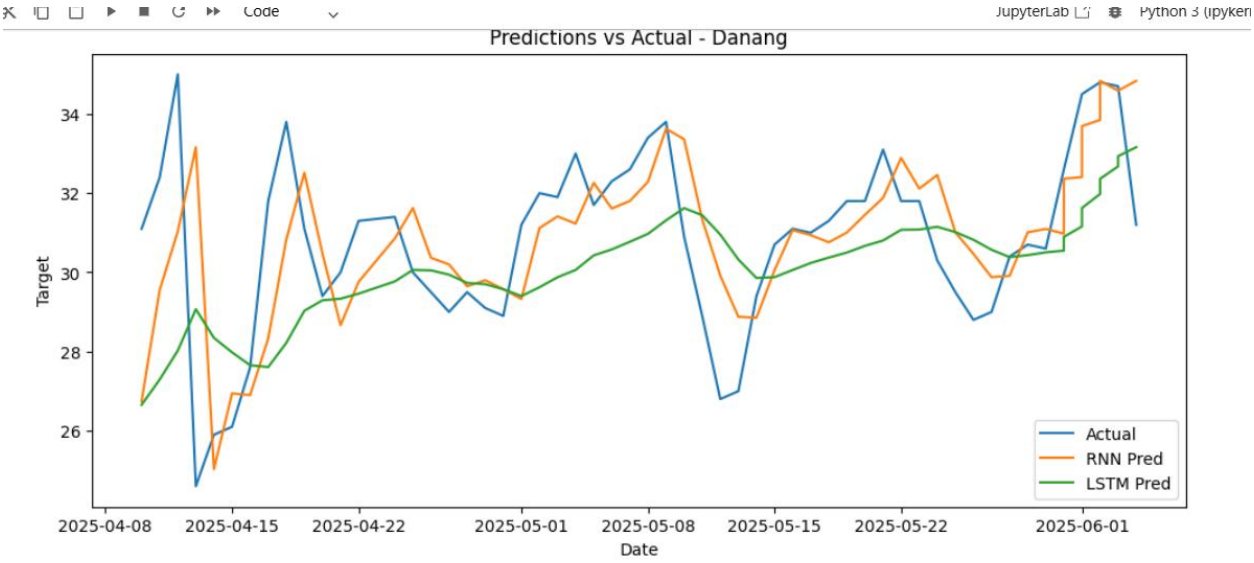

```
Danang features: ['day.maxtemp_c', 'day.maxtemp_f', 'day.mintemp_c', 'day.mintemp_f']
```

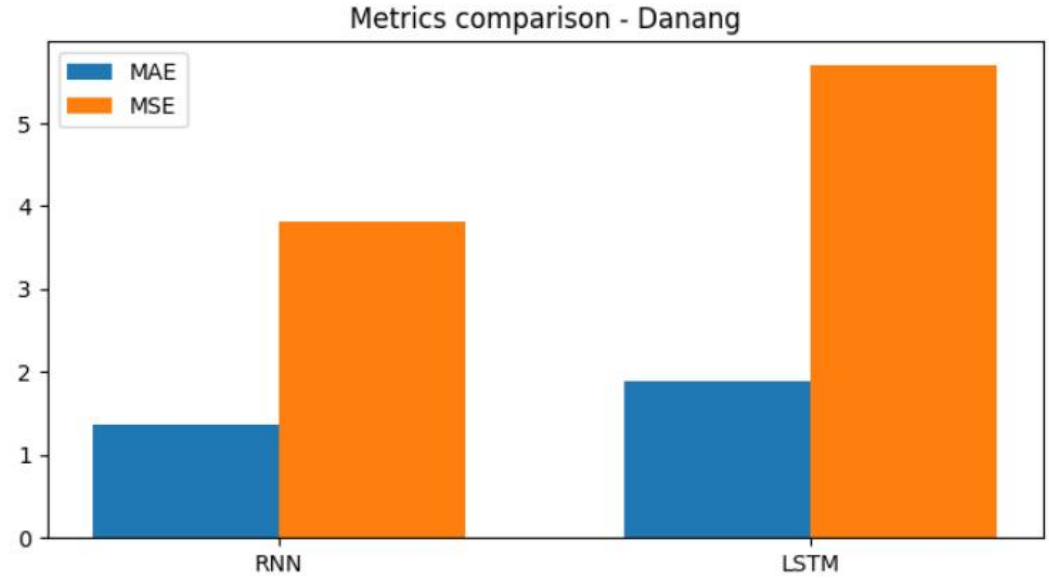
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 400ms/step
Metrics for Danang:
{'RNN': {'MAE': 1.367069606053627, 'MSE': 3.820015213995866, 'RMSE': 1.954485920644062, 'R2': 0.296502213004611
5, 'MSE': 5.700256906339712, 'RMSE': 2.387521079768661, 'R2': -0.04976496015586451}}
```



Epoch



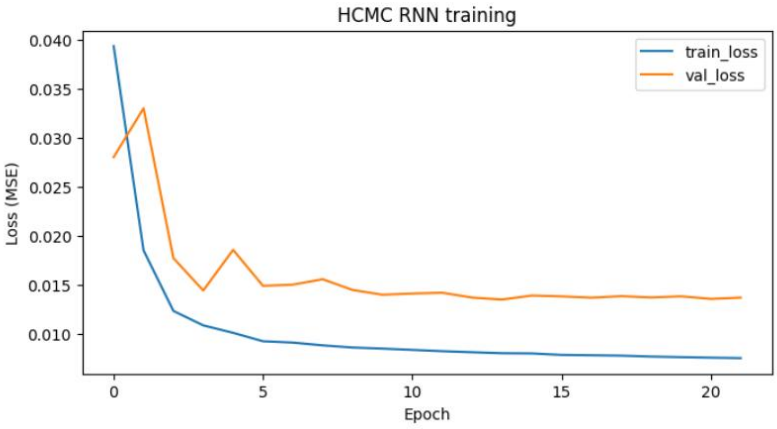




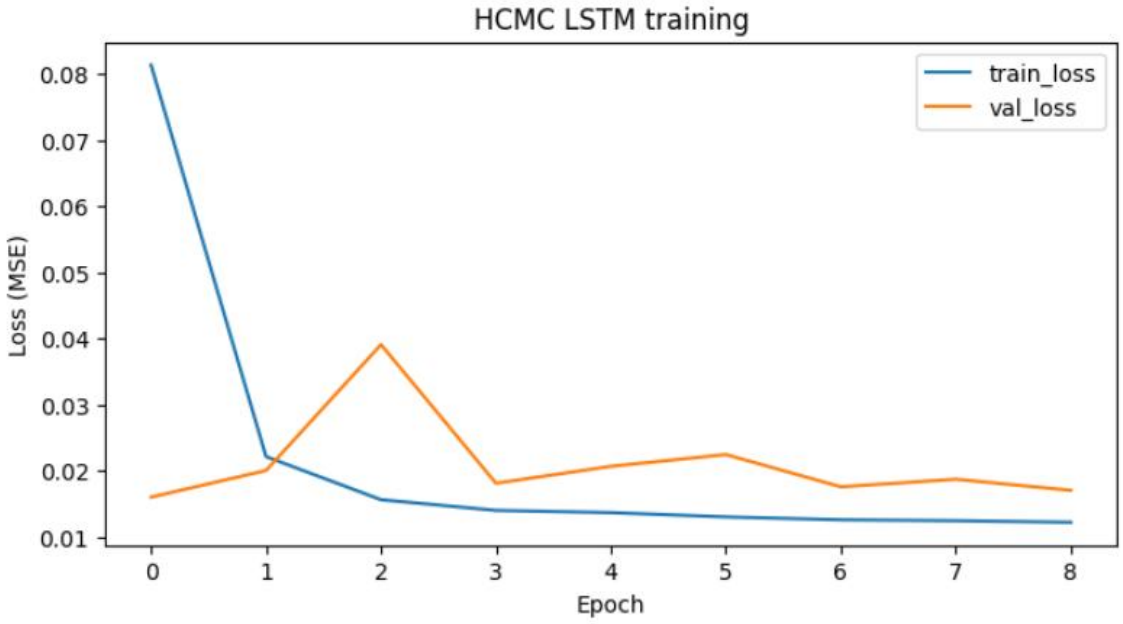

```
HCMC features: ['day.maxtemp_c', 'day.maxtemp_f', 'day.mintemp_c', 'day.mintemp_f']
```

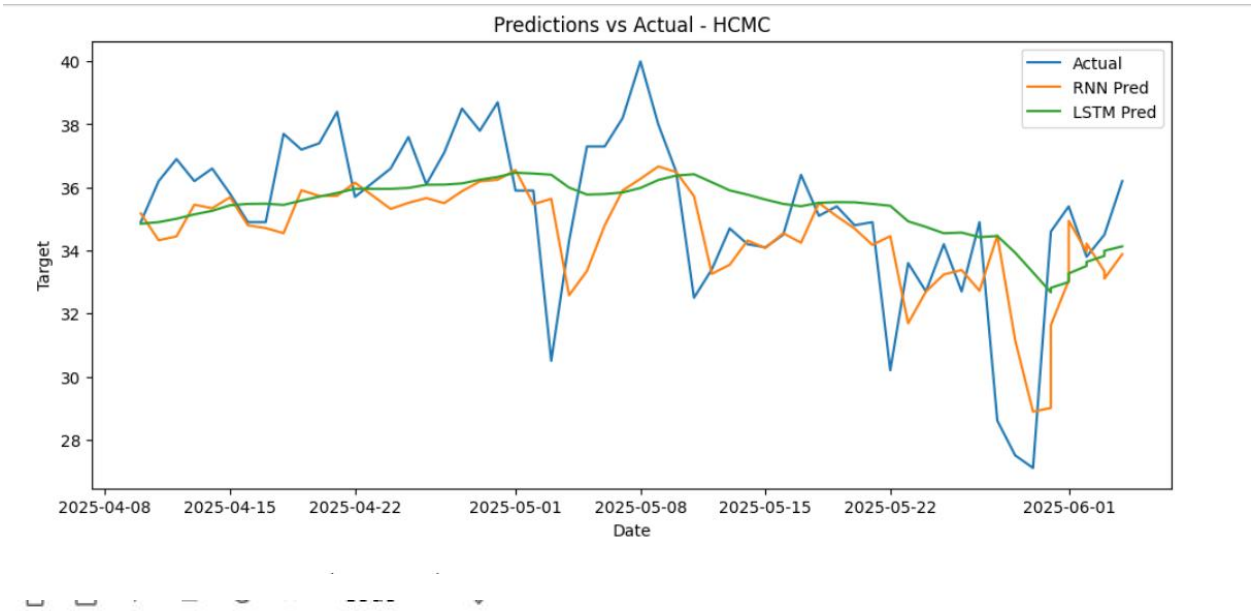
```
C:\Users\nyqua\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\rnn\rnn.py:199
```

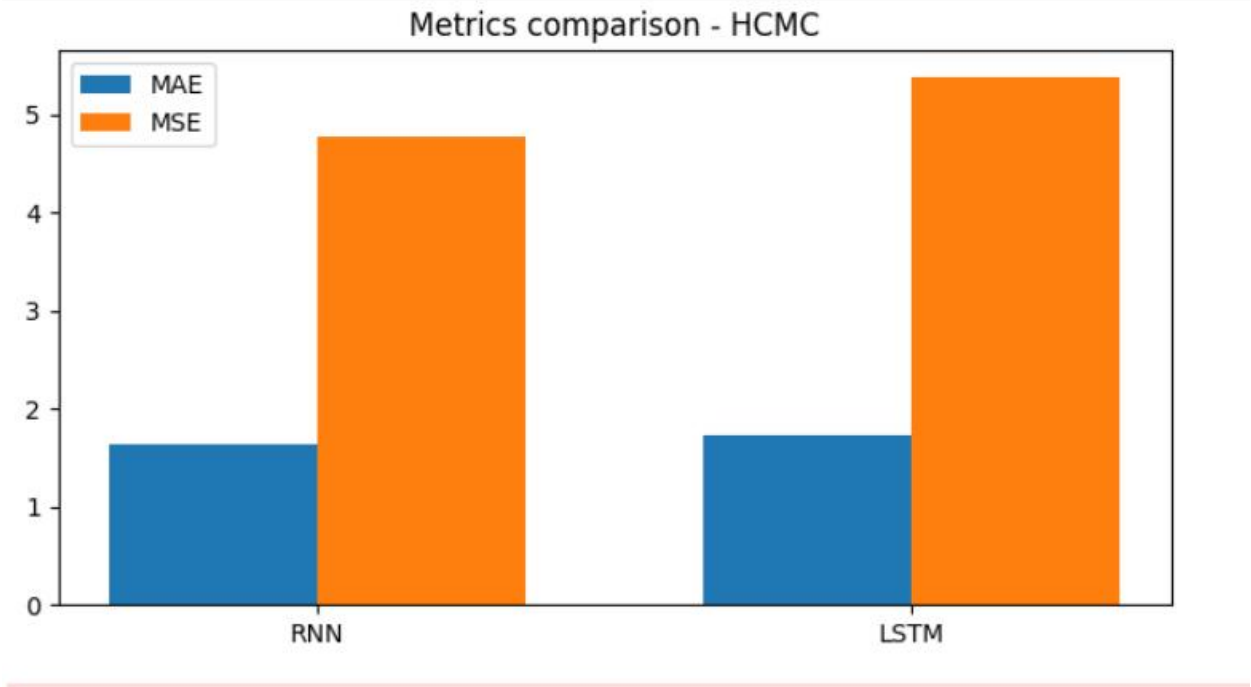
Metrics for HCMC:
{'RNN': {'MAE': 1.6373853070250994, 'MSE': 4.767727863119957, 'RMSE': 2.1835127348197347, 'R2': 0.26111652667089735}, 'LSTM': {'MAE': 1.7233009677822297, 'MSE': 5.380131384058844, 'RMSE': 2.3195110226206825, 'R2': 0.16620866833223202}}



Epocn




Predictions vs Actual - HCMC
Actual
RNN Pred
LSTM Pred
Target
Date
40
38
36
34
32
30
28
2025-04-08
2025-04-15
2025-04-22
2025-05-01
2025-05-08
2025-05-15
2025-05-22
2025-06-01


Metrics comparison - HCMC
MAE
MSE
5
4
3
2
1
0
RNN
LSTM

```
We recommend using instead the native Keras format, e.g. `model.save('my_model.
```

```
=== Summary comparison across cities ===
     city   rnn_MAE    rnn_MSE  rnn_RMSE    rnn_R2  lstm_MAE   lstm_MSE  \
0   Hanoi  2.576708  10.189694  3.192130  0.319954  2.752221  11.182637
1  Danang  1.367070   3.820015  1.954486  0.296502  1.892202   5.700257
2    HCMC  1.637385   4.767728  2.183513  0.261117  1.723301   5.380131

   lstm_RMSE   lstm_R2
0   3.344045  0.253686
1   2.387521 -0.049765
2   2.319511  0.166209
Saved models for Hanoi -> model_Hanoi_rnn.h5, model_Hanoi_lstm.h5
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ke
We recommend using instead the native Keras format, e.g. `model.save('my_model.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ke
We recommend using instead the native Keras format, e.g. `model.save('my_model.
```

```
Saved models for Danang -> model_Danang_rnn.h5, model_Danang_lstm.h5
Saved models for HCMC -> model_HCMC_rnn.h5, model_HCMC_lstm.h5
```