Enhancing Sequential Data Analysis Using the RVT Block Structure

Integrating Convolution, Self-Attention, and LSTM

By - Sai Sirisha Vadapalli

Roadmap

Introduction, Challenges, and Evolution

Understanding RVT and Its Components

Code Implementation and Implications

Introduction, Challenges, and Evolution

Introduction to Sequential Visual Data

- Sequential visual data consists of a series of images or frames presented in a specific order.
- Each image follows the previous one, creating a sequence that captures changes over time.

Real- Time Examples:

- Live Video Streams: Instagram Stories, TikTok Live
- Video Conferencing: Zoom Meetings, Microsoft Teams
- Tesla Dataset: Camera Feeds, LiDAR and Radar Reading, Sensor Fusion

The Challenges of Sequential Visual Data

The challenge with sequences of images and videos is understanding how things change over time and figuring out the connections between different moments.

Some of the challenges are:

- Understanding Time-Dependent Changes
- Temporal Patterns and Context
- Complex Dependencies
- Variable Sequence Lengths
- Real-World Applications

Evolution of Approaches for Sequential Visual Data

- RNN's: Recurrent Neural Networks (RNNs) were widely used to analyze sequential visual data.
- RNNs could capture temporal patterns and dependencies in sequences.
- Rise of Transformers: Transformers gained prominence in natural language processing for capturing long-range dependencies.
- The success led to exploring their potential in handling sequential visual data.
- But they have limitation.

Integration of Transformers and RNNs in Sequential Visual Data

- The success of Transformers in language tasks sparked interest in adapting them to sequential visual data, like images and videos.
- Concurrently, Recurrent Neural Networks (RNNs) excelled in modeling sequential data, capturing temporal patterns over time
- The idea of combining Transformers' global context understanding with RNNs' temporal pattern recognition emerged as a solution.
- Recurrent Vision Transformers (RVT) came into the picture as a novel architecture that bridges the strengths of both RNNs and Transformers.

Understanding RVT and Its Components

Recurrent Vision Transformer – Introduction and Significance

RVT combines two powerful concepts: Transformers and Recurrent Neural Networks (RNNs).

It's designed for enhancing image understanding and sequential modeling.

This architecture is particularly useful for tasks involving time-dependent visual data.

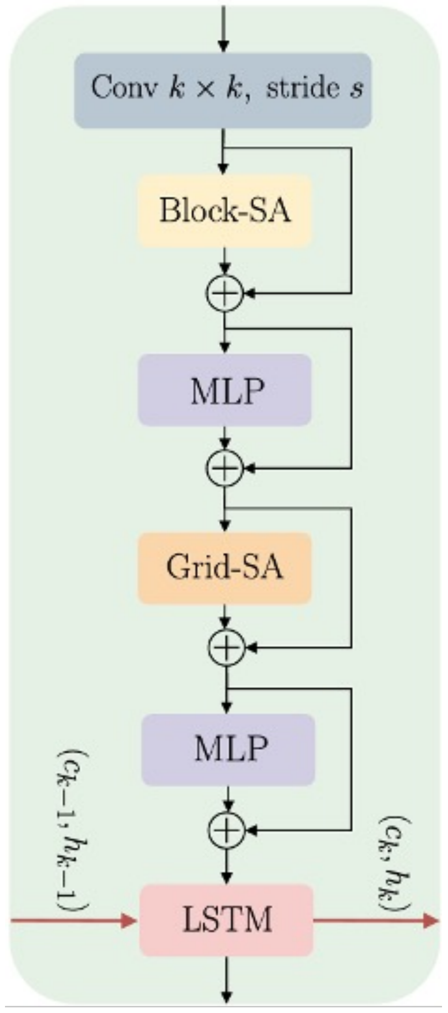
Why Combine Transformers and RNNs in RVT?

- Combining Transformers and Recurrent Neural Networks (RNNs) in Recurrent Vision Transformers (RVT) leverages their strengths for handling sequence-based visual data challenges.
- Strengths:
  1. Enhanced Sequential Understanding
  2. Global and Local Context
  3. Complex Temporal Dependencies
  4. Robustness to Sequence Lengths Improved Generalization
  5. Innovative Solutions
- This combination enables RVT to understand complex patterns and temporal dynamics in sequential visual data

RVT Block Structure

- The Recurrent Vision Transformer (RVT) Block is a fundamental building block that combines attention mechanisms and LSTM processing for sequential visual data

- <u>**Key Components:**</u>
  - Block Self-Attention
  - Grid Self-Attention
  - MLP Block
  - LSTM Block



Block Self- Attention

- The Block Self-Attention mechanism focuses on understanding relationships between different positions within a local region of the input feature map
- Block SA operates within a specific spatial neighborhood to capture contextual information effectively.

Key Steps in Block Self- Attention

- ❑ **Convolutional Feature Extraction:**
  - ❑ Input is convolved through a **conv2d_input** layer, extracting local patterns.
- ❑ **Feature Refinement with Convolutional Layers and Sigmoid:**
  - ❑ Resulting feature map undergoes two 3x3 convolutions.
  - ❑ Convolutional layers uncover spatial relationships, enriched by a sigmoid activation.
- ❑ **Attention Map Generation:**
  - ❑ Sigmoid output forms an attention map.
  - ❑ Map reflects position importance within the local context.
- ❑ **Feature Modulation:**
  - ❑ Attention map modulates the original feature map.
  - ❑ Relevant positions are amplified, less relevant ones suppressed.

Grid Self-Attention

The **GridSelfAttention** class is designed to capture **long-range dependencies** within input data, such as images or feature maps.

It achieves this by utilizing a specialized form of convolution called a **dilated convolution.**

The output of the dilated convolution is passed through a **ReLU (Rectified Linear Unit) activation function**, which introduces non-linearity and helps in learning complex patterns and relationships.

Grid Self-Attention

- **Class Definition**: Define the GridSelfAttention class as a subclass of tf.keras.layers.Layer.
- **Initialization**: In the constructor, specify the number of channels for the convolutional operation.
- **Convolution with Dilatio**n: Initialize a Conv2D layer with a dilation rate to capture long-range dependencies.
- **ReLU Activation**: Apply the ReLU activation function to the convolutional output, enhancing non-linearity and feature extraction.
- **Call Method**: Implement the call method to pass input data through the dilated convolution and ReLU activation.

Multi-Layer Perceptron

- The MLPBlock (Multi-Layer Perceptron Block) plays a role in introducing non-linear transformations to the input data.
- It consists of a series of fully connected layers (also known as dense layers), where each neuron is connected to all neurons in the previous layer.
- ReLU (Rectified Linear Unit) activation functions are applied to the outputs of these dense layers, introducing non-linearity and allowing the network to learn complex patterns.
- The final layer of the MLPBlock reduces the output dimensions, often leading to a higher-level representation of the input data.

Multi-Layer Perceptron

- **Class Definition**: Define the **MLPBlock** class as a subclass of **tf.keras.layers.Layer**.
- **Initialization**: In the constructor, provide the list of hidden units and the number of output channels.
- **Dense Layers**: Create a list of Dense layers with ReLU activation for each specified number of hidden units.
- **Output Layer**: Add a final Dense layer to reduce the output dimensions to the desired number of output channels.
- **Call Method**: Implement the **call** method to sequentially pass input through hidden layers and the final output layer.

Long- Short Term Memory

- The MLPBlock (Multi-Layer Perceptron Block) plays a role in introducing non-linear transformations to the input data.
- It consists of a series of fully connected layers (also known as dense layers), where each neuron is connected to all neurons in the previous layer.
- ReLU (Rectified Linear Unit) activation functions are applied to the outputs of these dense layers, introducing non-linearity and allowing the network to learn complex patterns.
- The final layer of the MLPBlock reduces the output dimensions, often leading to a higher-level representation of the input data.

Long- Short Term Memory

- **Class Definition**: Define the **RVTBlock** class including LSTM as a recurrent layer.
- **Initialization**: Specify the input channels, output channels, and LSTM hidden size.
- **LSTM Layer**: Initialize the LSTM layer with the specified hidden size.
- **Call Method**: In the **call** method, reshape the input data for LSTM processing.
- **LSTM Processing**: Pass the reshaped input through the LSTM layer with initial hidden and cell states.
- **Hidden and Cell States**: Retrieve the LSTM outputs, new hidden states, and new cell states for the next time step.

Code Implementation and Implications

Setting the Stage: Libraries and Dataset

Libraries Used:

- matplotlib.pyplot: Employed for visualization creation, including plots of attention and feature maps, as well as output analyses.
- tensorflow: Leveraged from TensorFlow, an open-source ML framework, for constructing neural network models with specialized layers, convolution operations, and LSTM cells.
- numpy: Utilized for numerical operations, specifically generating synthetic input data for the purpose of demonstrating the RVT architecture.

Hyperparameter Configuration and Creating RVTBlock Instance

- Hyperparameters are pivotal settings that fundamentally shape the behavior of a machine learning model. In the context of our codebase, we meticulously configure the subsequent hyperparameters:
- output_channels: A determinant of feature representation depth, thereby influencing model capacity and complexity.
- lstm_hidden_size: Dictates the quantity of hidden units within the LSTM layer, pivotal for capturing sequential dependencies.
- These hyperparameter configurations are of paramount significance, as they enable fine-tuning the Recurrent Vision Transformer (RVT) model to align with the specific task demands.

Creating RVTBlock Instance

- The following instantiation suffices for this encapsulation:
- **rvt_block = RVTBlock(input_channels=3, output_channels=output_channels, lstm_hidden_size=lstm_hidden_size)**
- **input_channels=3**: Explicitly specifies the quantity of input channels, pertinent in scenarios such as RGB image processing.
- **output_channels=output_channels**: Defines the output channel dimensionality for layers within the RVTBlock.
- **lstm_hidden_size=lstm_hidden_size**: Configures the count of hidden units nestled within the LSTM layer.

Generating Random Data and Sequential Processing Step

<u>Random Data Generation:</u>

- num_samples: 2
- sequence_length: 2
- input_height: 3
- input_width: 3
- input_channels: 1

- Data Shape: (num_samples, sequence_length, input_height, input_width, input_channels).

Sequential Processing Step

- **Time Steps Loop**:
  - Iterate over each time step in the sequence.
- **Convolutional Feature Extraction**:
  - Obtain features through convolutional layers.
  - Extract features using rvt_block.conv2d_input(random_input[:, t]).
- **Block-Level Self-Attention**:
  - Apply BlockSelfAttention mechanism.
  - Compute x_block_sa = rvt_block.block_sa(x_conv).
- **Attention Map Evolution**:
  - Visualize attention maps for insight.
  - Store mean_attention_map from x_block_sa.

Sequential Processing Step

- **Grid Self-Attention**:
  - Apply GridSelfAttention mechanism.
  - Compute x_grid_sa = rvt_block.grid_sa(x_conv).
- **MLP and LSTM Processing**:
  - Process through MLPBlock and LSTM layers.
  - Calculate x_mlp = rvt_block.mlp_block(x_block_sa).
  - Combine features x_combined = x_mlp + x_grid_sa.
- **Reshaping for LSTM**:
  - Reshape for LSTM input.
  - Convert x_combined into x_reshaped for LSTM.

Sequential Processing Step

LSTM Processing:

- Process through LSTM layer.
- Obtain LSTM output, hidden states, and new LSTM states.
- Store lstm_out, new_lstm_states_h, new_lstm_states_c.

This sequence showcases how input data evolves through various processing steps in the RVTBlock architecture

.

Convolutional Feature Extraction and Attention Mechanisms

- Convolutional Feature Extraction:
- The conv2d_input method applies a convolutional layer to the input data.
- x_conv = rvt_block.conv2d_input(random_input[:, t])
- Convolutional layers extract features using kernels that slide over the input.
- They capture spatial hierarchies and identify patterns in the data.

Convolutional Feature Extraction and Attention Mechanisms

- Convolutional Feature Extraction:
- The conv2d_input method applies a convolutional layer to the input data.
- x_conv = rvt_block.conv2d_input(random_input[:, t])
- Convolutional layers extract features using kernels that slide over the input.
- They capture spatial hierarchies and identify patterns in the data.

Attention Mechanisms

- **Block Self-Attention**
- The BlockSelfAttention layer calculates attention weights using convolutions and a sigmoid activation. x_block_sa = rvt_block.block_sa(x_conv)
- Self-attention enhances representations by learning interdependencies within the same feature map.
- The sigmoid activation scales attention weights between 0 and 1.

Attention Mechanisms

- Grid Self-Attention:
- The GridSelfAttention layer employs dilated convolutions to capture long-range dependencies.
-

```
x_grid_sa = rvt_block.grid_sa(x_conv)
```

- Grid self-attention expands the receptive field, enabling the model to consider broader contexts.
- Dilated convolutions effectively incorporate information from distant positions.

MLP and LSTM Processing

- MLP : **Multi-Layer Perceptron (MLP) Block**
- **The MLPBlock layer consists of densely connected layers.**
- x_mlp = rvt_block.mlp_block(x_block_sa)
- MLP layers learn complex, nonlinear patterns in the data.
- Hidden units progressively reduce dimensions to extract high-level features.

MLP and LSTM Processing

- LSTM Block Integration:
- LSTM captures sequential patterns and maintains hidden states.
- lstm_out, new_lstm_states_h, new_lstm_states_c = rvt_block.lstm_block(x_reshaped, initial_state=lstm_states)
- LSTM takes reshaped, combined features and learns temporal dependencies.
- Hidden states store learned context and influence future predictions.

Attention Evolution and Output Analysis

Output Analysis

1. **Convolutional Feature Extraction**:

   - The Conv2D layer extracts essential features from the input data.

   - *Visualization*: Features are represented using the 'viridis' color map. Darker shades highlight lower values, while brighter shades indicate higher values.

2. **Block Self-Attention**:

   - This mechanism applies self-attention to extracted features.

   - *Visualization*: Attention maps use the 'hot' color map. Warmer colors (reds) indicate higher attention values, while cooler colors (blues) indicate lower attention values.

3. **Evolution of Attention Maps**:

   - Attention maps evolve as the model processes the sequence.

   - *Visualization*: 'hot' color map tracks attention changes. Warmer colors signify higher attention areas, and cooler colors signify lower attention areas.

Output Analysis

4. **Grid Self-Attention**:

   - Grid Self-Attention captures long-range dependencies.

   - *Visualization*: 'hot' color map highlights spatial relationships. Warmer colors denote areas of higher attention.

5. **Combined Output Analysis**:

   - Combined output merges features from Block Self-Attention and Grid Self-Attention.

   - *Visualization*: 'gray' color map showcases individual features. Lighter shades represent higher values, and darker shades represent lower values.

6. **LSTM Outputs and Hidden States Visualization**:

   - LSTM processes sequences over time steps.

   - *Visualization*: Green (LSTM outputs) and purple (hidden states) colors distinguish outputs in the plot.

Future Exploration

Model Behavior Analysis: Delve into how the RVT model performs when exposed to various hyperparameters and different types of data domains.

Visualizing Deeper Insights: Experiment with alternative visualization methods to gain a richer understanding of the attention mechanisms and their impact.

Performance Benchmarking: Evaluate the RVT architecture's effectiveness by benchmarking it against established datasets, allowing a comparison of its outcomes with prevailing techniques.

Key Takeaways

Reshaping Computer Vision: The Recurrent Vision Transformer (RVT) introduces a powerful fusion of transformer and recurrent layers, revolutionizing the handling of intricate visual data.

Unveiling Modern Neural Dynamics: Through this exploration, we've uncovered the intricate dynamics of the RVT model, contributing to a better grasp of cutting-edge neural architecture.

THANK YOU