1.PAY ATTENTION

a) auto-regressive self-attention.
As we can know, to achieve self-attention, we have these formulas:

$y_i = \sum_{j=1}^{n} W_{ij} x_j$, $w_{ij} = x_i^T x_j$, $W_{ij} = softmax(w_{ij})$

Since every token only attends to its own position and all previous positions in auto-regressive self-attention. Assume that i is the index of the input elements, $i = 1, \ldots, n$. For the token i, it needs to consider i values.

Thus, the number of dot products $N = 1 + 2 + \cdots + n = \frac{n(1+n)}{2}$

Here is the n grid and shade/mark the squares that depict the calculated attention scores. The cell with 'X' means that this cells needs to be calculated.

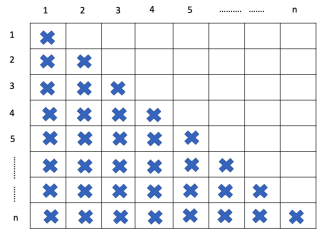

Figure 1. grid of auto-regressive self-attention

b) strided self-attention.
As we can know, to achieve self-attention, we have these formulas:

$y_i = \sum_{j=1}^{n} W_{ij} x_j$, $w_{ij} = x_i^T x_j$, $W_{ij} = softmax(w_{ij})$

Since every token attends to at most t positions prior to it, plus itself in strided self-attention. Assume that i is the index of the input elements, $i = 1, \ldots, n$.

For the token i where $i \leq t+1$, it needs to consider i values.
For the token i where $i > t+1$, it needs to consider $t+1$ values.

Thus, the number of dot products $N = \sum_{i=1}^{t+1} i + \sum_{i=t+2}^{n} t + 1 = \frac{(1+t+1)(t+1)}{2} + (n-t-1)(t+1) = \frac{t^2+3t+2}{2} + nt + n - t^2 - 2t - 1 = -\frac{1}{2}t^2 - \frac{1}{2}t + nt + n$

Here is the n grid and shade/mark the squares that depict the calculated attention scores. The cell with 'X' means that this cells needs to be calculated. In this grid, we assume t is 3.

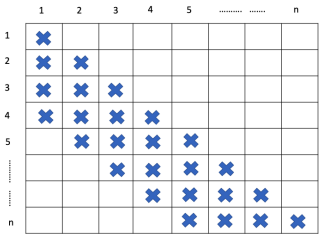

Figure 2. grid of strided self-attention

c) windowed self-attention.
As we can know, to achieve self-attention, we have these formulas:

$$y_i = \sum_{j=1}^{n} W_{ij} x_j,\ w_{ij} = x_i^T x_j,\ W_{ij} = softmax(w_{ij})$$

Since the n tokens are partitioned into windows of size w (assume w divides n), and every token attends to all positions within its window and prior to it, plus itself in windowed self-attention. Assume that i is the index of the input elements, $i = 1, \ldots, n$.

As we can image, in each window, the n grid is the same as the n grid that we draw in question (a).

Thus, the number of dot products in each window $N = 1 + 2 + \cdots + w = \frac{w(1+w)}{2}$. And the total number of dot products $N = \frac{n}{w} \times \frac{w(1+w)}{2} = \frac{n(1+w)}{2}$

Here is the n grid and shade/mark the squares that depict the calculated attention scores. The cell with 'X' means that this cells needs to be calculated.

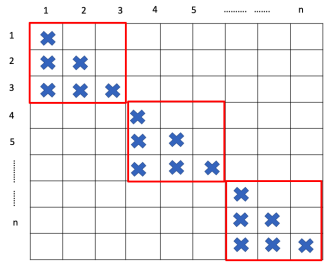

Figure 3. grid of windowed self-attention

2.RNNS FOR IMAGES.

a) the number of trainable parameters.
As we known, the equation for the hidden unit is given as follows:

$$h^{i,j} = \sigma(W_{in} x^{i,j} + W_{left} h^{i-1,j} + W_{top} h^{i,j-1})$$

Assume that the input grid is $n \times n$, each input $x^{i,j}$ is a c-channel vector, and the hidden state $h^{i,j-1}$ has dimension h.

Thus, the size of each input $x^{i,j}$ is $1 \times c$.
For input $x^{0,0}$, $h^{0,0} = \sigma(W_{in} x^{i,j})$ and $h^{0,0}$ has dimension h, which means the size of $h^{0,0}$ is $1 \times h$, we can know that the size of W_{in} is $h \times c$
And since the size of $h^{0,0}$ is $1 \times h$ and the size of $W_{left} h^{i-1,j}$ should also be $1 \times h$, we can know that the size of W_{left} is $h \times h$. So does the W_{top}.

As we known, the parameters here would be shared in the RNN. Therefore, the number of trainable parameters $N = hc + 2h^2$

b) the number of arithmetic operations.

First, lets calculate the arithmetic operations in $W_{in} x^{i,j}$:
For each hidden unit, it needs to do c times multiple operations and $c - 1$ times add operations. There are h hidden units. Thus, the number of multiple operations is hc and the number of add operations is $h(c-1)$

Then. lets calculate the arithmetic operations in $W_{left}h^{i-1,j}$.
For each hidden unit, it needs to do h times multiple operations and $h-1$ times add operations. There are h hidden units. Thus, the number of multiple operations is h^2 and the number of add operations is $h(h-1)$.
And the case in $W_{top}h^{i-1,j}$ is the same.

Therefore, for each time we calculate $h^{i,j}$, we need to do $hc+2h^2 = o(h^2)$ multiple operations. For add operations, besides those we counted above, we still need to sum three matrix up and each of them has h values. Thus, we need to do $h(c-1)+2h(h-1)+2h = O(h^2)$ add operations. Totally $hc+2h^2+h(c-1)+2h(h-1)+2h = O(h^2)$ arithmetic operations for calculating each $h^{i,j}$.

Also there are cases that the number operations is different.
For example, for $h^{i,j}$ when $i \geq 1 and\, j = 0$ or $j \geq 1 and\, i = 0$, we need to do $hc+h^2 = o(h^2)$ multiple operations and $h(c-1)+h(h-1)+h = O(h^2)$ add operations. However, the big-Oh notation are the same, thus in the following calculation we don't need to consider and measure these cases separately.
For $h^{0,0}$, we need to do $hc = o(h)$ multiple operations and $h(c-1) = O(h)$ add operations. However, we only have one case of this, so when we have $n > 1$ we can ignore this.

Considering input grid is $n \times n$, the total number of arithmetic operations is $O(h^2) \times n^2 = O(n^2h^2)$, where the numbers of add operations and multiple operations are the same as $O(n^2h^2)$.

c) regular convnet verse RNN.

CNNs employ filters within convolutional layers to transform data. Whereas, RNNs reuse activation functions from other data points in the sequence to generate the next output in a series.

Advantage: This RNN architecture can interpret temporal information such as videos (which are essentially a sequence of individual images) while CNN cannot. As this RNN architecture can analyze the current image with the images that it has already analyzed. Thus, it can explore the sequence relationship/pattern better. RNN is more used for cases where the data contains temporal properties, such as a time series. Similarly, where the data is context sensitive, the function of memory provided by the feedback loops is critical for adequate performance. while CNN cannot do this.
Moreover, this RNN architecture can take the inputs that have variable length.

Disadvantage: This RNN architecture is not as general as CNN on computer vision. CNN can be used on most of the images tasks such as image classification, segmentation or object detection.
CNNs can better learn the position and scale of features in a variety of images, making them especially good at the classification of hierarchical or spatial data and the extraction of unlabelled features, while this RNN architecture cannot do this.
And This RNN architecture might require more hyperparameters and supervision compring to CNN, which using small size of filters to do convolution on images.