Conv lstm - <https://www.kaggle.com/code/kcostya/convlstm-convolutional-lstm-network-tutorial>

Conv3d- <https://keras.io/api/layers/convolution_layers/convolution3d/>

A generator is a function that returns an iterable set of items, one at a time, in a special way using the yield statement instead of return. When a generator function is called, it returns a generator object without even beginning execution of the function. When next() is called on the generator object, the function starts executing until it hits the yield statement, which returns the yielded value and pauses the function’s state. Calling next() again resumes the function’s execution right after the yield.

In deep learning, especially when dealing with large datasets, generators are a way to efficiently load and preprocess data during model training. A generator is a type of iterable, like a list or tuple, but unlike lists, generators do not store their contents in memory. Instead, they generate items on the fly, which makes them particularly useful for handling large datasets that do not fit into memory all at once.

### Understanding Parameters in **Conv3D** and **ConvLSTM2D** Layers

Let's break down the computation of the number of parameters in Conv3D and ConvLSTM2D layers step-by-step for better understanding.

### Conv3D (3D Convolutional Layer)

In a Conv3D layer, parameters include the weights and biases used in the convolutional filters. Here’s how we compute them:

\*\*Definitions:\*\*

- \*\*Wc\*\*: Number of weights in the convolutional layer.

- \*\*Bc\*\*: Number of biases in the convolutional layer.

- \*\*Pc\*\*: Total number of parameters in the convolutional layer (Pc = **Wc + Bc**).

- \*\*K\*\*: Size (width) of the kernels used in the convolutional layer.

- \*\*N\*\*: Number of kernels (or filters).

- \*\*C\*\*: Number of channels in the input image.

3\*3 - kernel size

3\*3 - no of input channels for a video

=81\*16

\*\*Given Data:\*\*

- Kernel size, \*\*K = **3x3** = 9\*\* (since we're using 3x3 kernels).

- Number of input channels, \*\*C = **3** (R, G, B) x **3** (frames) = 9\*\*.

- Number of kernels (or feature maps), \*\*N = 16\*\*.

**Kernel** - **w\*h\*d\*f** ->**3**\***3**\***3**\***3 - 81**

Kernel filters - 16

\*\*Computing Weights (Wc):\*\*

- Each kernel has dimensions \*\*K x K x C\*\*, resulting in each kernel having \*\*9 x 9 = 81\*\* weights.

* Kernel size (3\*3\*3\*3\*no of kernel filters)
* (3\*3\*3\*3\*16)
* X dim\*ydim\* no of frames\* channel rgb \* no of filters

- There are \*\*N = 16\*\* such kernels.

- Thus, the total number of weights, \*\***Wc** = 9 x 9 x 16 = **1296\***\*.

K - square matrix - **3\*3 = 9**

D- 3 \* 3 => 3\*3\*3\*3 = **81 (overall size of kernel filter)**

**81\*16 (no of filters) = Wc**

**Every kernel fillter will have biases sigma(wtx+b) =>16 biases**

Square matrix of size 3.3 => **9.3**.**3** = 9.9= 81 elements present in the weight matrix \* no of kernel filters taken into first conv layer = 81.16

K - 3\*3 (square matrix) \* 3 \*3\*16 -> weights utilized for trainable parameters

Biases -

\*\*Computing Biases (Bc):\*\*

- Each kernel has one bias term.

- Thus, for \*\*N = **16**\*\* kernels, we have \*\*Bc = 16\*\*.

\*\*Total Parameters (Pc):\*\*

- \*\*Pc = Wc + Bc = 1296 + 16 = **1312\***\*.

### ConvLSTM2D (Convolutional Long Short-Term Memory Layer)

A ConvLSTM2D layer combines convolutional operations with LSTM’s gating mechanisms. Here’s how we compute the parameters:

\*\*Definitions:\*\*

- \*\*K\*\*: Kernel size.

- \*\*Cin\*\*: Number of input channels.

- \*\*Cout\*\*: Number of output channels.

- LSTM has three gates and 4 matricess

\*\*Given Data:\*\*

- Kernel size, \*\*K = 3\*\*.

- Number of input channels, \*\*Cin = 3\*\*.

- Number of output channels (feature maps), \*\*Cout = 16\*\*.

\*\*Computing Parameters:\*\*

- The total number of parameters for ConvLSTM2D is given by:

**[K × K × (Cin + Cout) × Cout+Cout]** \* 4

LSTM -3 gates but we use 4 matrices

- Breaking it down:

**- Each gate has a kernel of size \*\*K x K x (Cin + Cout)\*\*.**

**- Each gate has \*\*Cout\*\* output channels.**

**- Thus, each gate’s weights = \*\*K x K x (Cin + Cout) x Cout\*\*.**

**Filtersize . input data . output data**

**- Each gate has \*\*Cout\*\* biases.**

Conv

Depth + output -> inpt to lstm

Output to lstm -

k\*k\*(cin+cout) \*cout (kernel weights) + cout (biases)

- For the given data:

- Weights per gate: \*\*3 x 3 x (3 + 16) x 16 = 3 x 3 x 19 x 16 = 2736\*\*.

- Biases per gate: \*\*16\*\*.

- Total per gate: \*\*2736 + 16 = **2752**\*\*. -RNN

- There are 4 matrices in an LSTM.

- Thus, total parameters: \*\*2752 x 4 = **11008**\*\*.

### Summary

- \*\*Conv3D Layer\*\*:

- Kernel size: 3x3 (9).

- Number of input channels: 9.

- Number of kernels: 16.

- Total parameters: \*\*1312\*\*.

- \*\*ConvLSTM2D Layer\*\*:

- Kernel size: 3x3.

- Number of input channels: 3.

- Number of output channels: 16.

- Four gates (input, output, forget, cell).

- Total parameters: \*\*11008\*\*.

By breaking down the calculations into these clear steps, you can see how the total number of parameters is derived for each type of layer. This understanding helps in designing and optimizing deep learning models effectively.

The ideal batch size for training a machine learning model depends on several factors, including model performance, computational resources, and training time. Here are some concise guidelines:

### Key Considerations

1. \*\*Model Performance\*\*:

- \*\*Smaller Batch Sizes\*\* (e.g., 32): Often lead to better generalization but slower training.

- \*\*Larger Batch Sizes\*\* (e.g., 128): Can speed up training but might require more epochs and can lead to overfitting.

2. \*\*Computational Resources\*\*:

- \*\*Memory Constraints\*\*: Larger batches need more memory.

- \*\*Efficiency\*\*: GPUs handle larger batches well due to parallelism.

3. \*\*Training Time\*\*:

- Larger batches can speed up training per epoch but might require fine-tuning of learning rates.

### Practical Tips

- \*\*Starting Point\*\*: Begin with a batch size of 32 or 64.

- \*\*Common Sizes\*\*: Use powers of 2 (e.g., 32, 64, 128, 256).

- \*\*Empirical Testing\*\*: Experiment with different sizes and monitor training/validation performance.

- \*\*Balance\*\*: Find a balance between efficient training and good generalization.

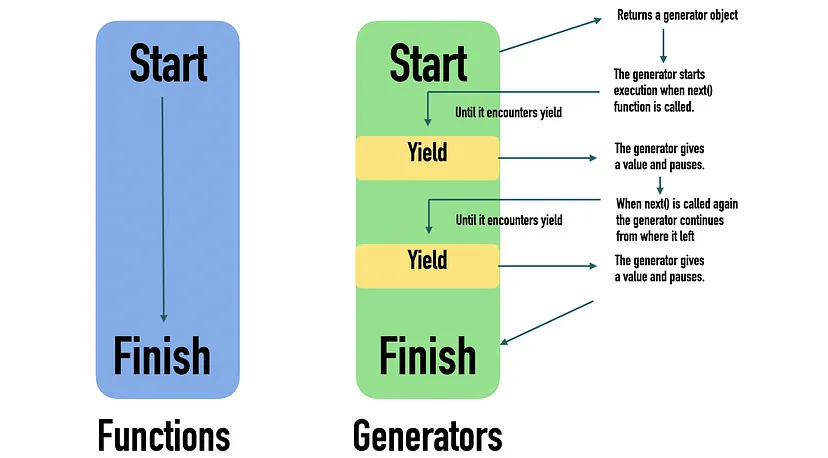
### Example

1. \*\*Start\*\*: With a batch size of 32.

2. \*\*Monitor\*\*: Training and validation performance.

3. \*\*Adjust\*\*: Increase if memory allows and training is stable; decrease if memory is constrained or if generalization suffers.

By following these guidelines and adjusting based on your specific needs and resources, you can determine the optimal batch size for your model.



In Python, yield is used to return from a function *without* destroying its variables. In a sense, yield pauses the execution of the function. When the function is invoked again, the execution continues from the last yield statement.

Using yield turns a function into a *generator*.

`ReduceLROnPlateau` is a callback function in Keras, a popular deep learning library, used during model training to dynamically adjust the learning rate based on the model's performance on a validation set.

Here's a breakdown of its functionality:

1. \*\*Monitoring Metric\*\*:

- The callback monitors a specified metric during training, typically the validation loss or validation accuracy.

2. \*\*Plateau Detection\*\*:

- It detects plateaus in the monitored metric, which indicate that the model's performance is no longer improving significantly.

3. \*\*Adjusting Learning Rate\*\*:

- When a plateau is detected (i.e., the monitored metric does not improve for a specified number of epochs), the learning rate is reduced by a certain factor. This reduction helps the optimizer navigate through the loss landscape more effectively and potentially find a better minima.

4. \*\*Patience\*\*:

- A parameter called "patience" determines the number of epochs with no improvement required before the learning rate is reduced. This prevents premature adjustments in cases where the model's performance fluctuates.

5. \*\*Minimum Learning Rate\*\*:

- Optionally, a minimum learning rate can be specified. Once the learning rate reaches this threshold, it will not be reduced further.

6. \*\*Verbosity\*\*:

- It provides verbosity options, allowing users to control the amount of output messages regarding the learning rate reduction.

In summary, `ReduceLROnPlateau` is a valuable tool for fine-tuning the learning process during model training, potentially leading to faster convergence and improved performance. It helps adapt the learning rate dynamically based on the observed behavior of the model on the validation set.