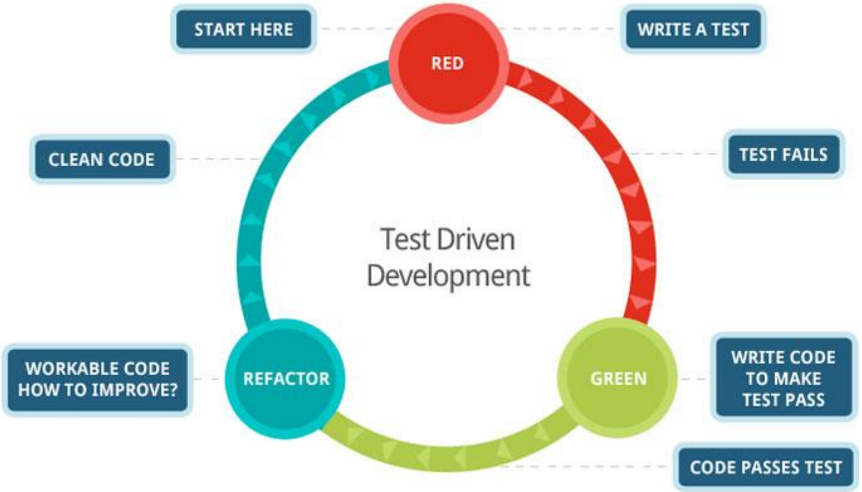
Convolutional Layer

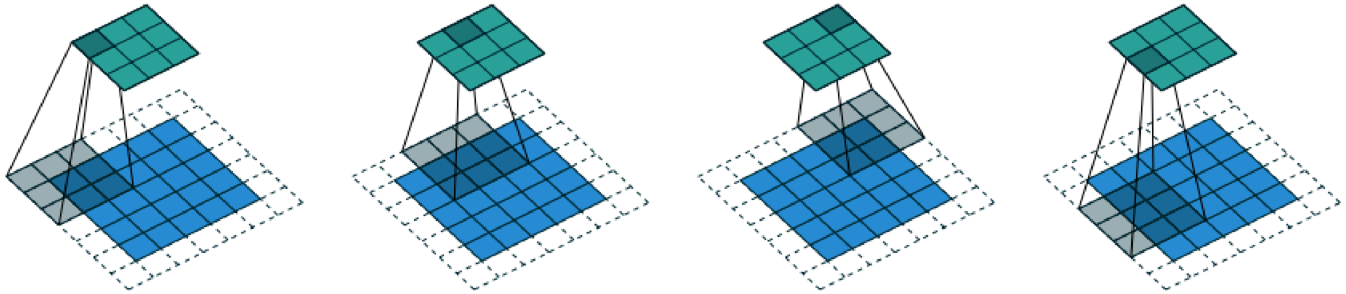
Implementation & Testing

Overview

- Convolutional Layer
- Unit testing
- Integration testing
- Future Work



Convolutional Layer



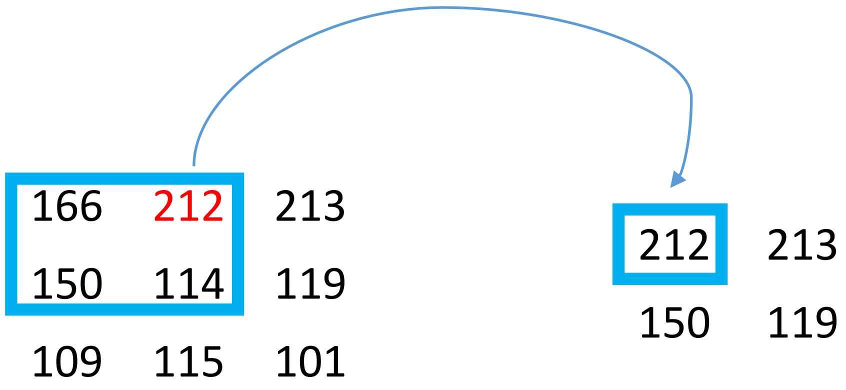
Convolving a 3x3 kernel over a 5x5 input, using ***s = 2*** *and* ***p = 1***

Extra Considerations

- Both the input volume and kernel are generally **3D**.
- Each layer includes **more than one kernel**.
- Stride and padding values **do not need to be equal** in the two dimensions.
- The kernel and input tensors are **not necessarily square**.

Unit testing

Testing a single operation in isolation



Test cases

- Different strides and padding.
- Non square kernels.
- Depth > 1.

The same testing strategy was employed for forward and backward propagation in all supported layer types.

Fun fact: Doing that for convolutional layer back-prop took me more time than implementing it!

Integration testing

Great! Each piece works, therefore the system works! **Right…?**

Hint: **Wrong**

Generally a system is more than the sum of its pieces.

Integration Testing – Approach 1

We could use the same strategy as in unit testing:

1. Create an input.
2. Create kernels for each layer.
3. **Compute the output of each layer on paper.**
4. **Compute the gradients for every layer on paper.**

I was actually stubborn enough to do that.

This would probably take me a month to compute and would include about 5 to 10 mistakes.

Enter finite difference computation!

Credit to: **Simon Pfreundschuh**

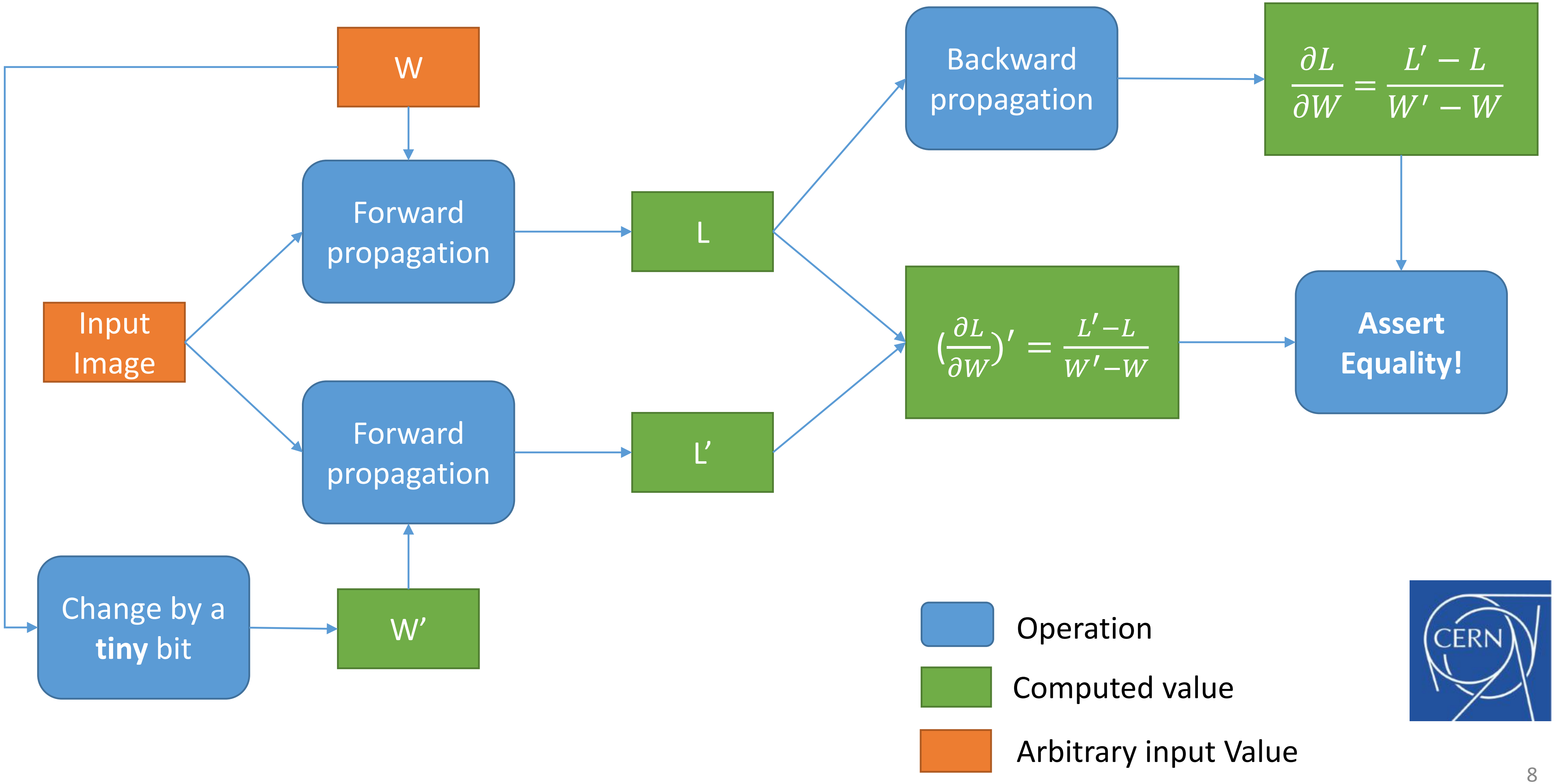
Let’s go back to the definition of $\frac{\partial L}{\partial W} = \lim_{dW \to 0} \left(\frac{L(W+dW) - L(W)}{dW} \right)$

Interpretation:

How much will the **Loss** change if I change ***W*** by **a tiny** bit?

Well, let’s do exactly that!

Finite difference comparison



Results & Future Work

- My implementation recently passed this test! Hurray!

- **Since its (supposedly) correct, it's now time to make it fast!**