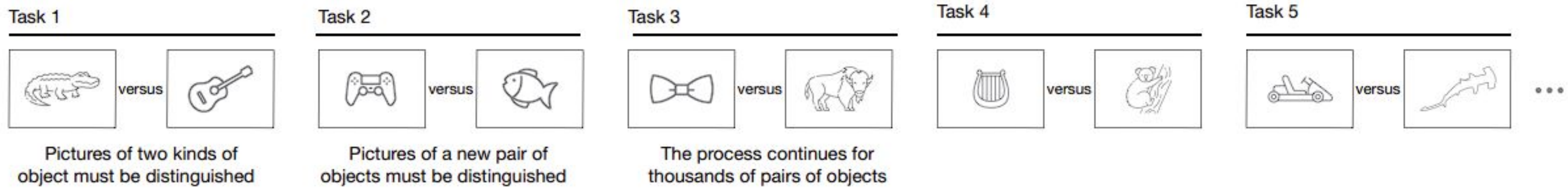
Continual Learning:

Plasticity Stability Dilemma in Computer Vision

Continual Learning: Setup

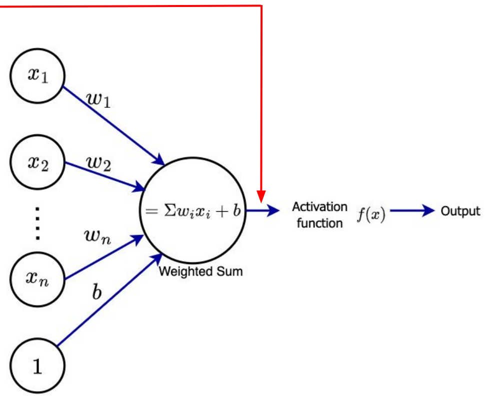
Problem Setting



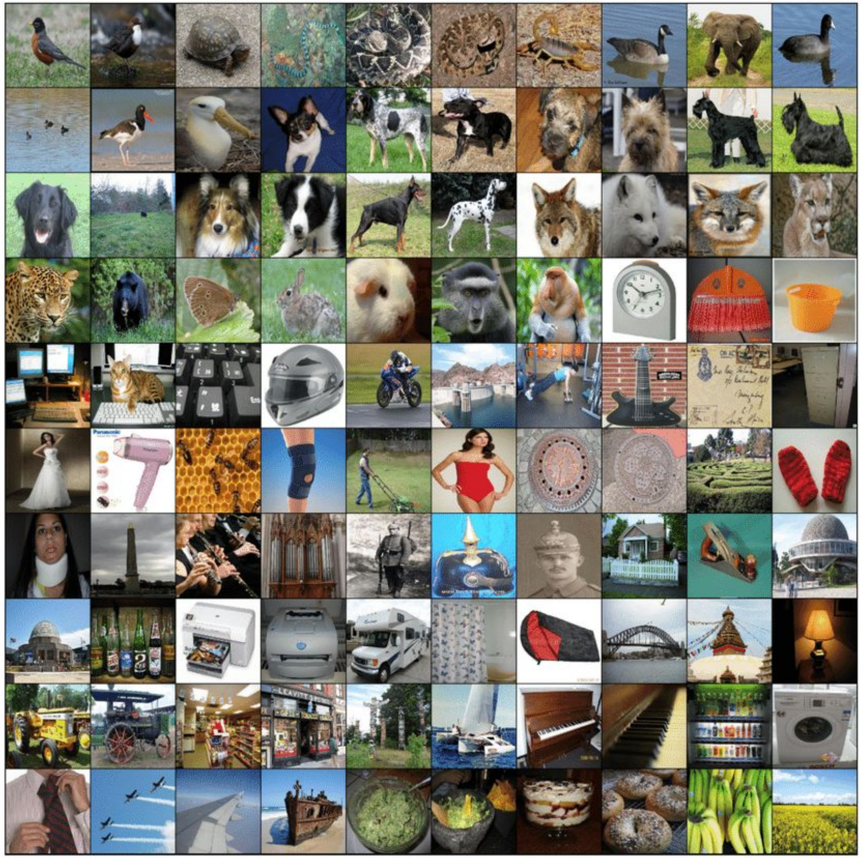
- Distinguish pairs Images from each other in sequence
  - 32x32 ImageNet
  - No Rehearsal
- Network: Input –> 3xConv –> 2xFC –> Output

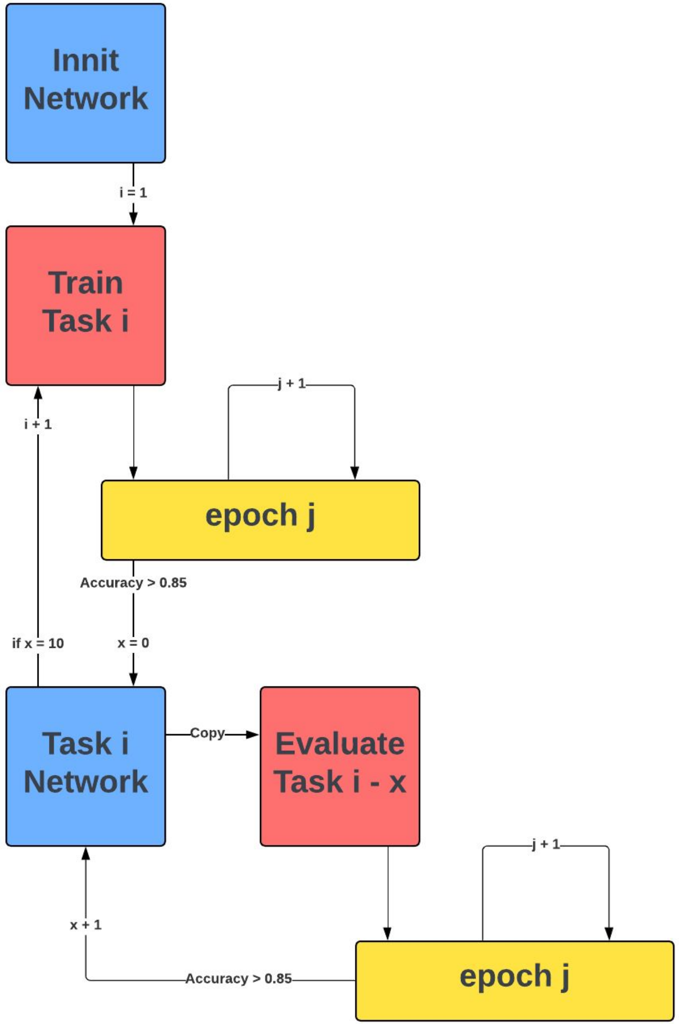
Metrics

- Plasticity:
  - Normally: Performance in Current Task after x Epochs
  - Adapted: Epochs to Reach a Performance of 85%
- Stability:
  - Normally: Performance in the past Tasks without additional Training
  - Adapted: Epochs to regain a Performance of 85% in the past Tasks
- Secondary
  - Preactivation Distribution



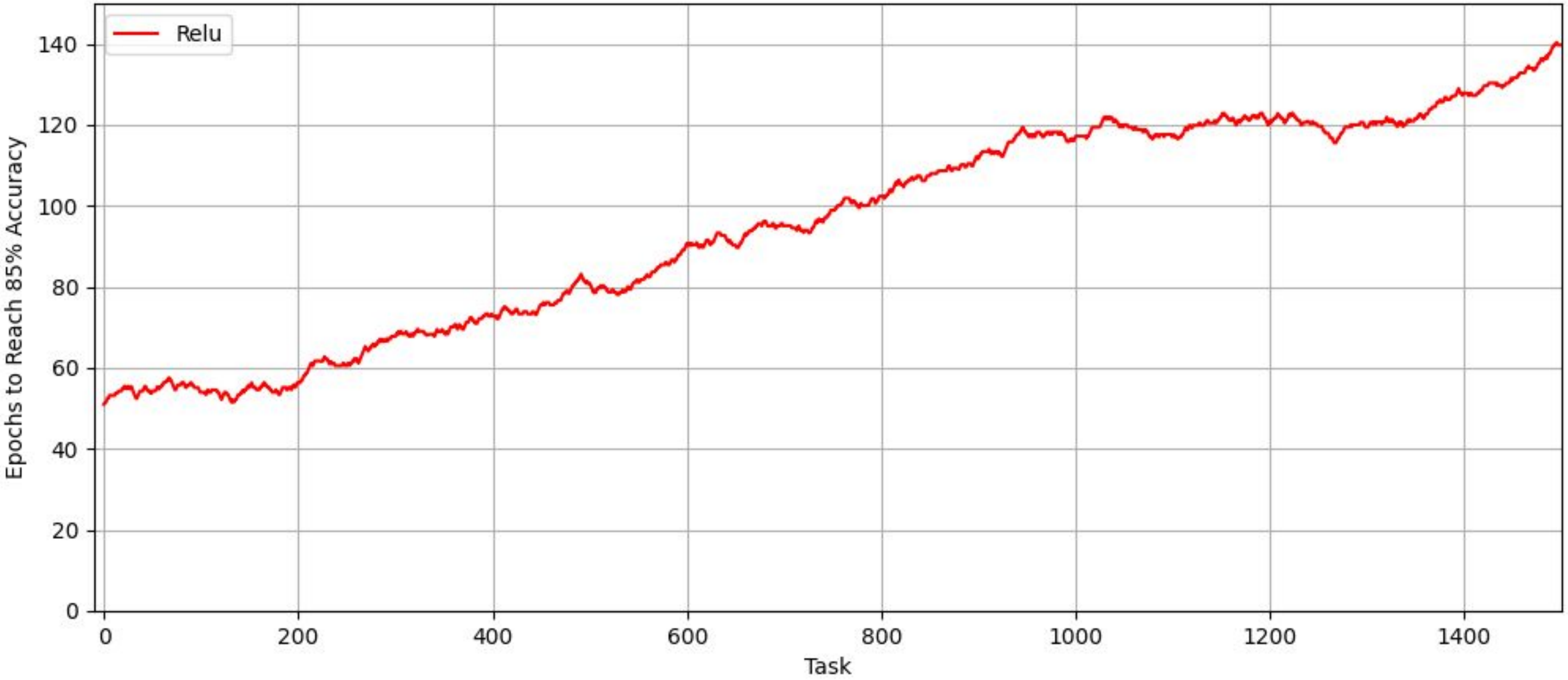
Flow Diagram of Training





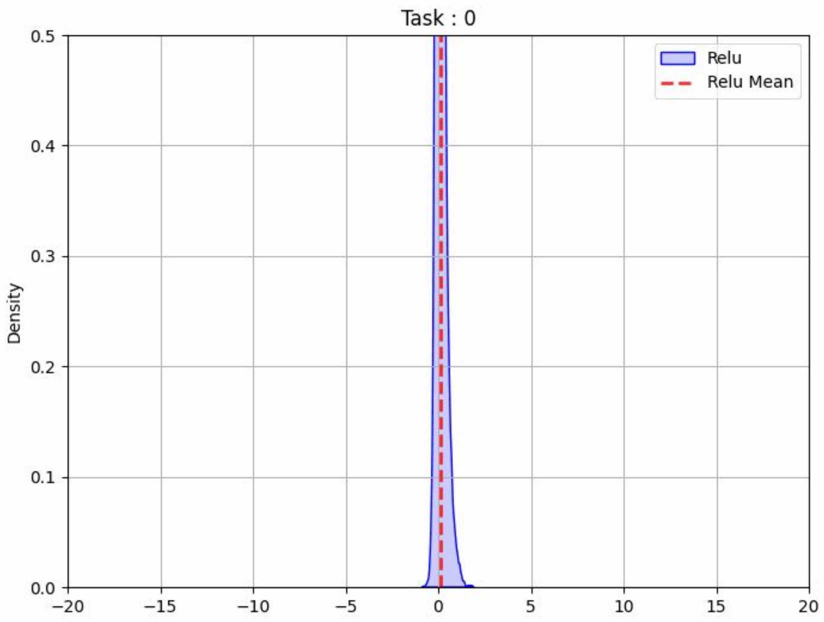
Continual Learning: Plasticity

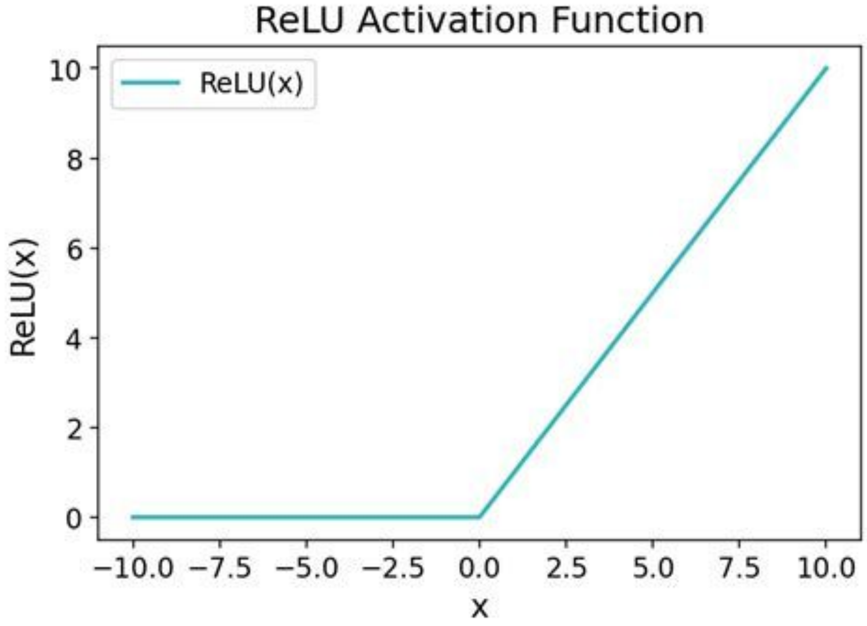
Performance of A Convolutional Network with a ReLu



Why do we lose plasticity?

- Relu has a preactivation shift to the left

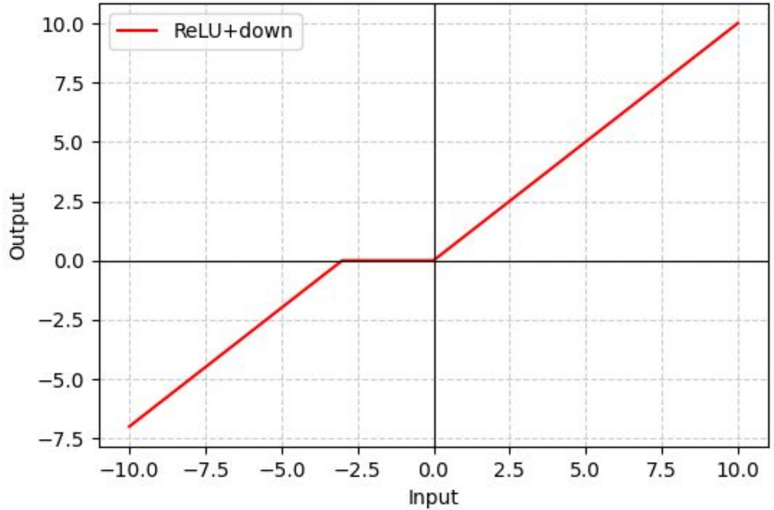




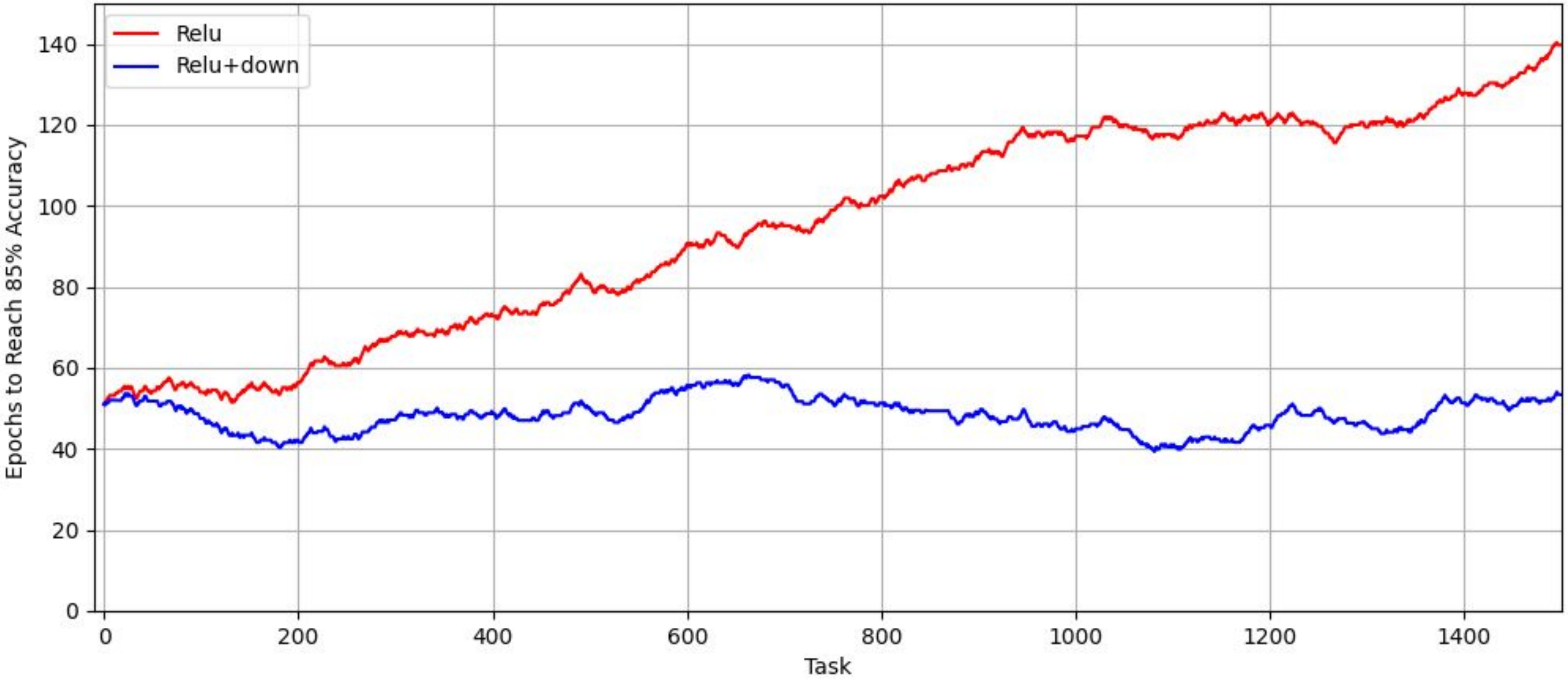
Using Relu+down to maintain Plasticity

- Adapting the Relu Function to prevent Preactivation Distribution Shift
- Achieving this by using Relu+down
- Properties
    - point symmetrical
    - easy to implement
    - more expressive than the standard relu

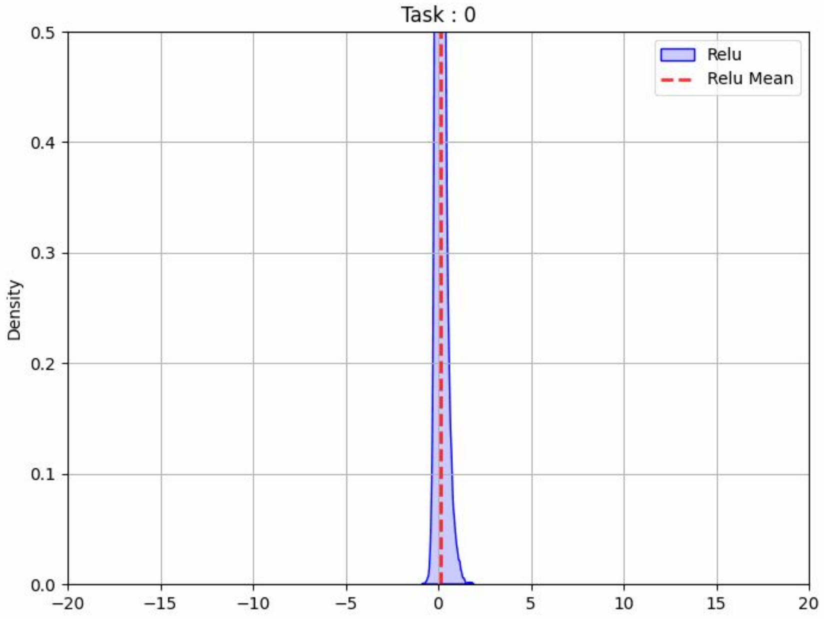
up

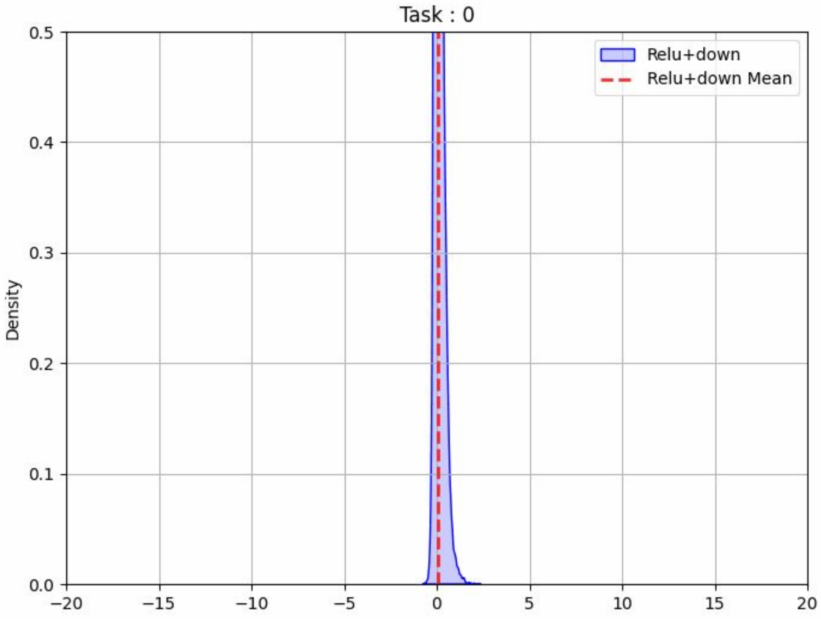


Performance of A Convolutional Network with a ReLu



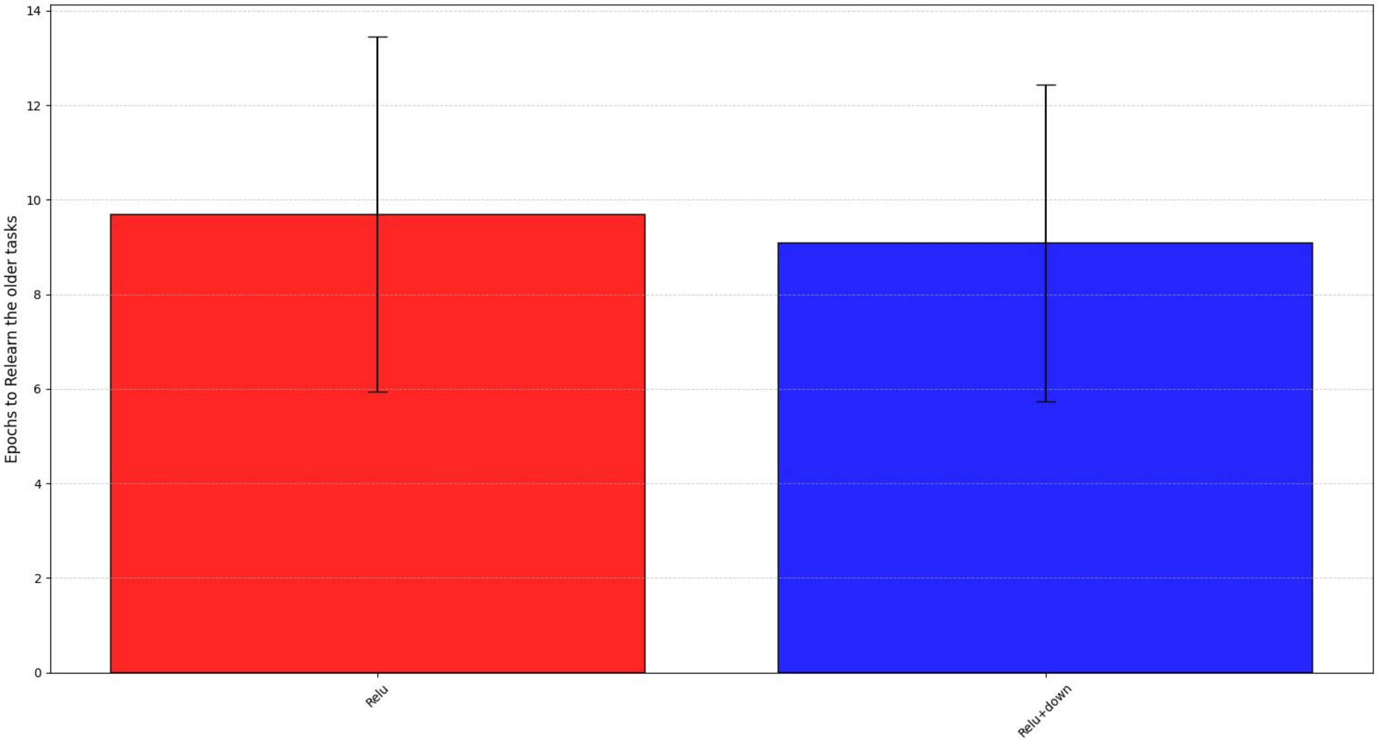
Preactivation Shift: Relu and Relu+down



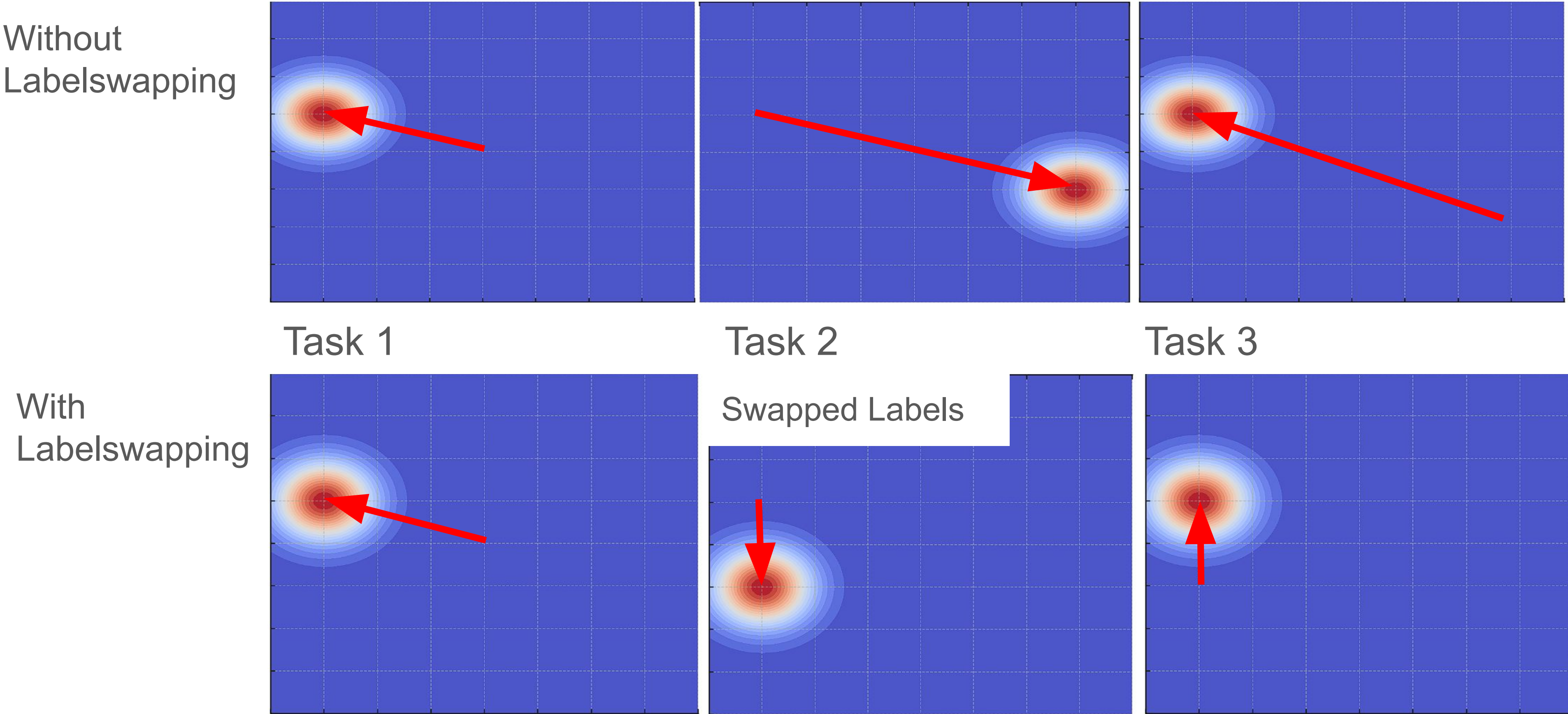


Continual Learning: Stability

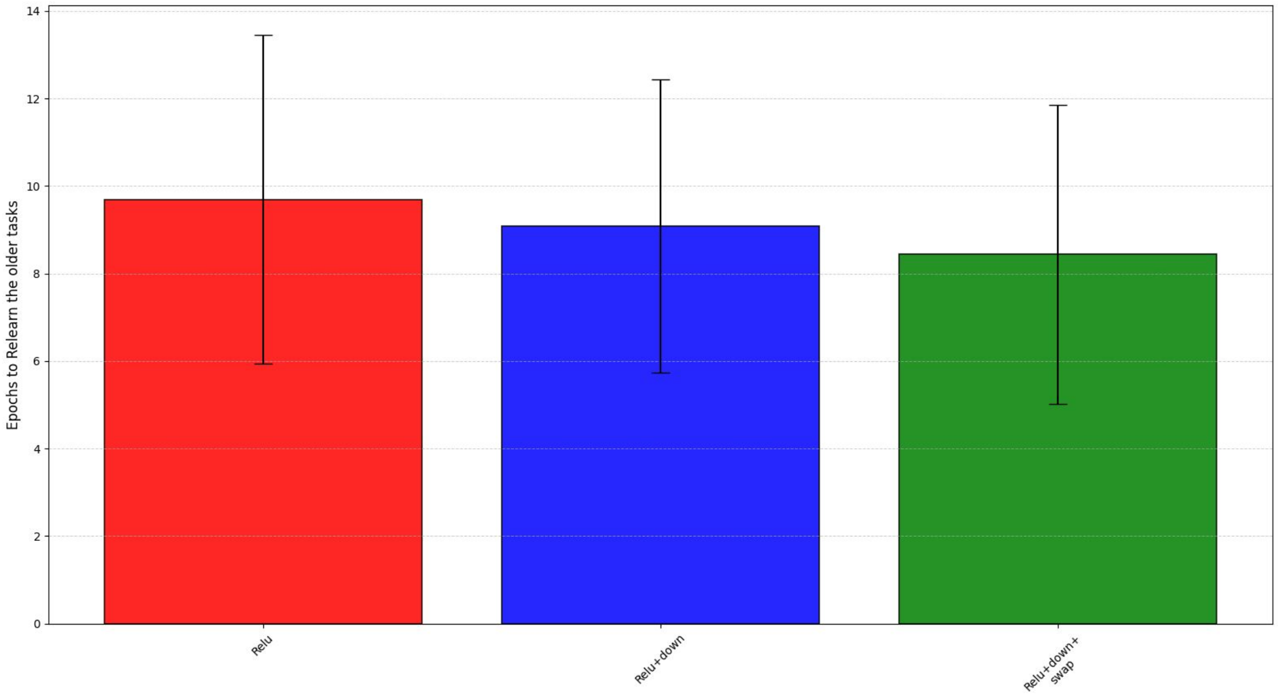
Performance of the Networks in Stability



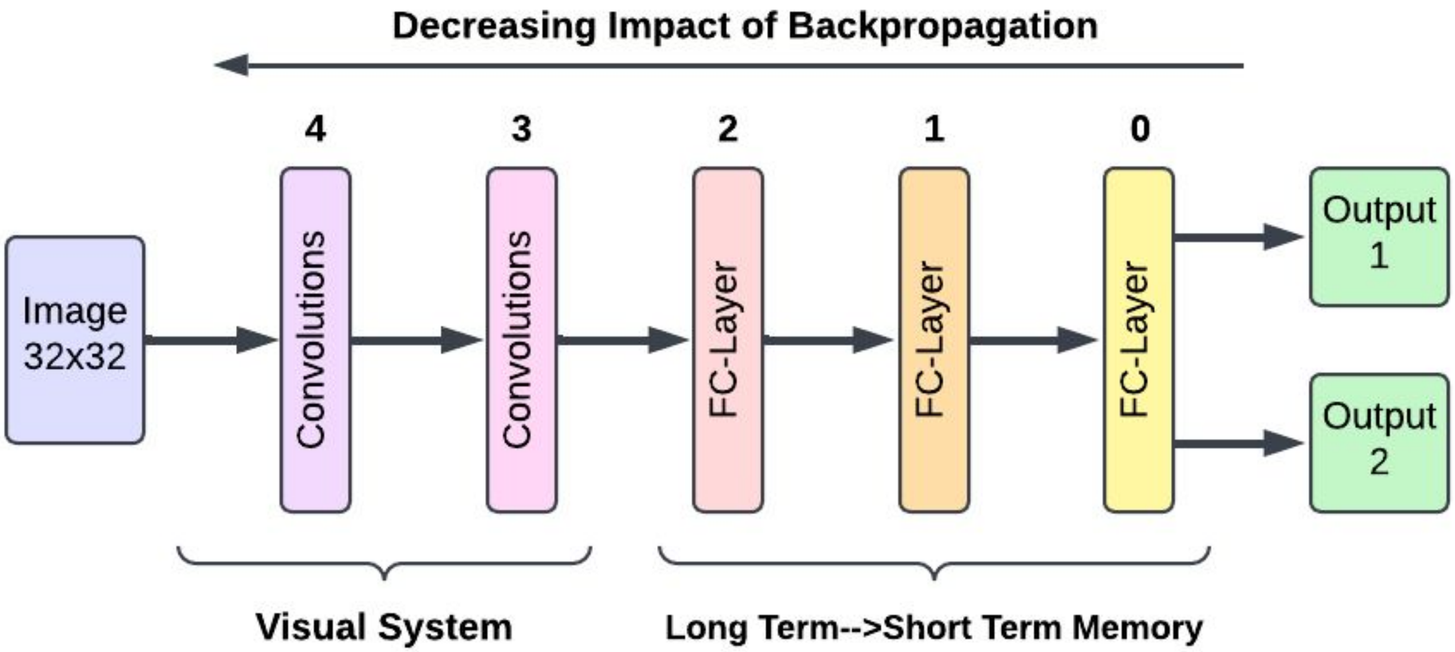
Intuition: Label Swapping



Performance of the Networks in Stability



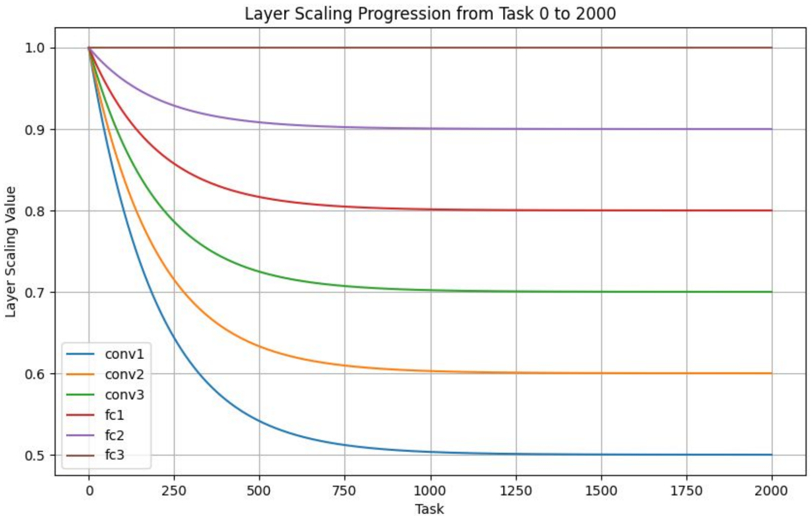
Intuition: Decreasing Backpropagation



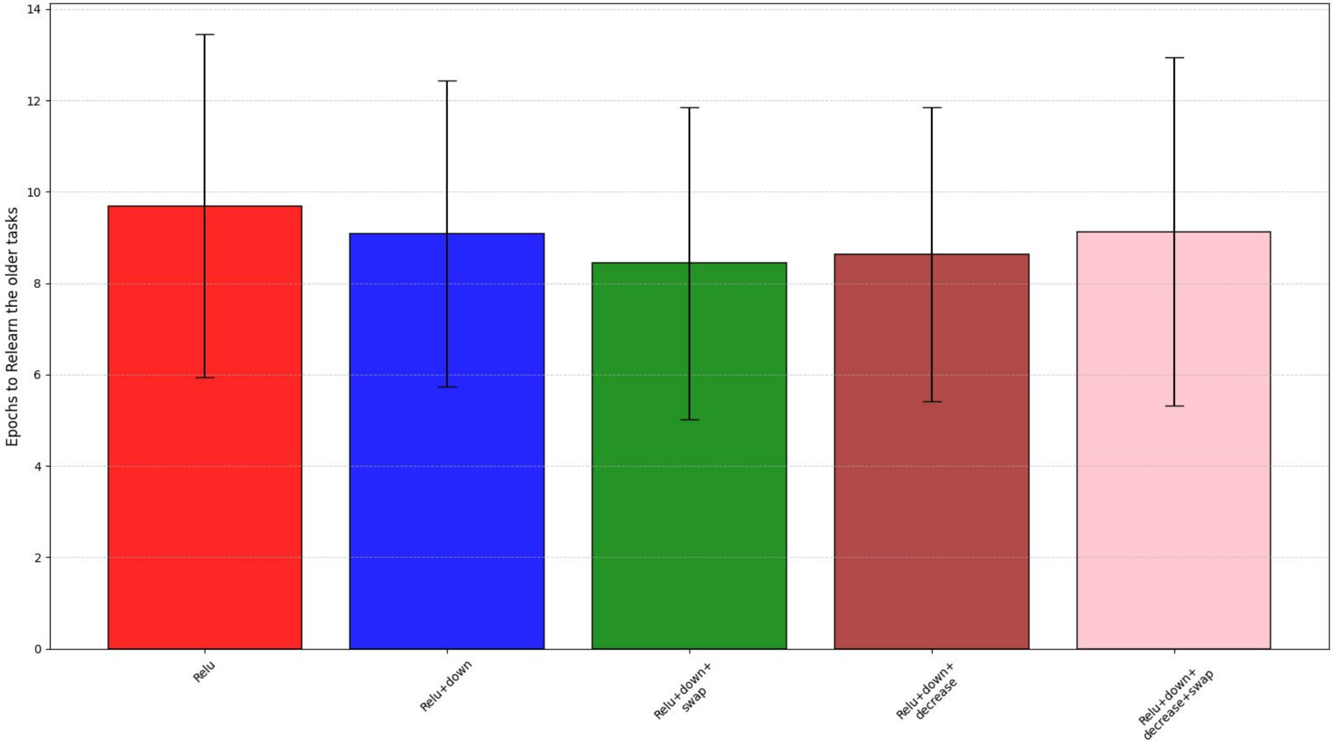
Intuition: Decreasing Backpropagation

These Values are finetuned! (To a certain extend)

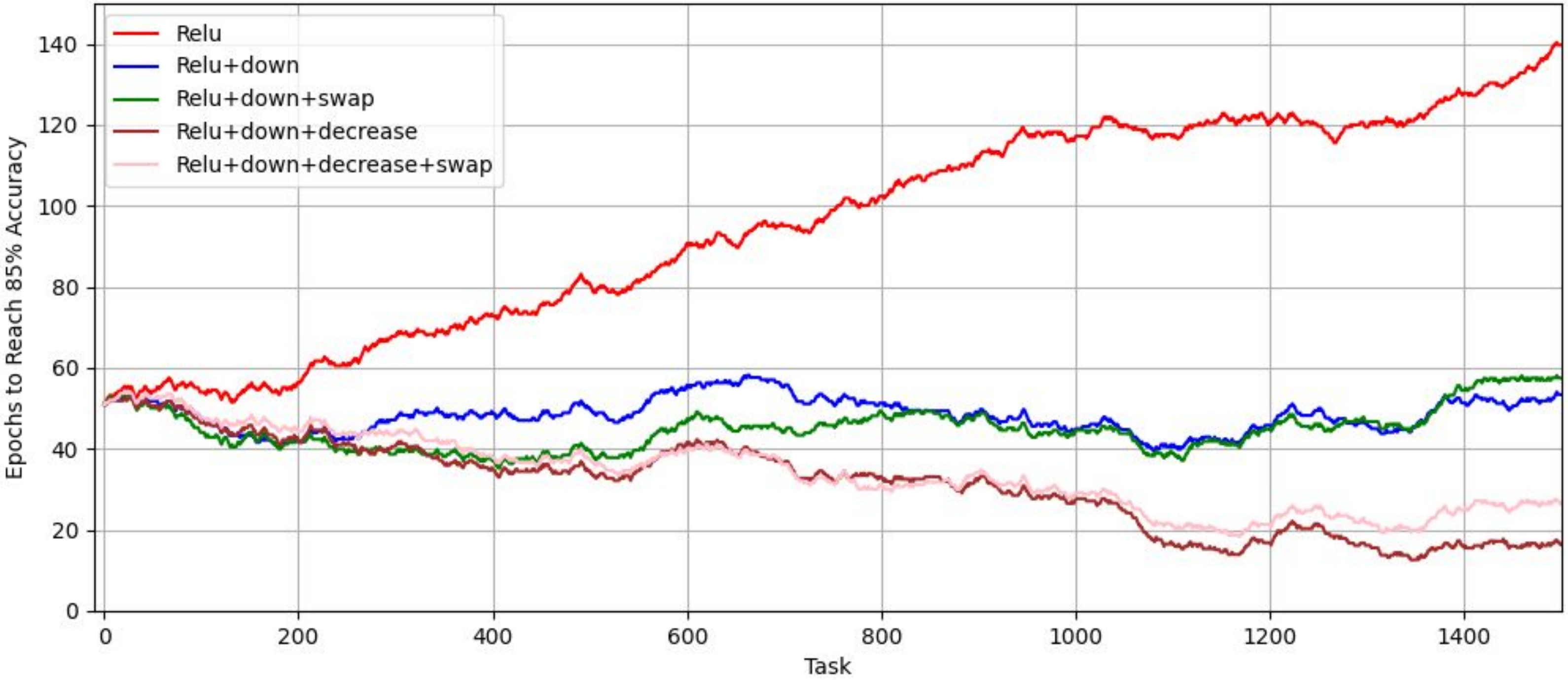
```
def learn(self, x, target,task):
    layer_scaling = {
        "conv1.weight": 0.5,
        "conv1.bias": 0.5,
        "conv2.weight": 0.6,
        "conv2.bias": 0.6,
        "conv3.weight": 0.7,
        "conv3.bias": 0.7,
        "fc1.weight": 0.8,
        "fc1.bias": 0.8,
        "fc2.weight": 0.9,
        "fc2.bias": 0.9,
        "fc3.weight": 1.0,
        "fc3.bias": 1.0
    }
    layer_scaling = {name: scale + (1 - scale)*1.005**-task for name, scale in layer_scaling.items()}
```



Performance of the Networks in Stability

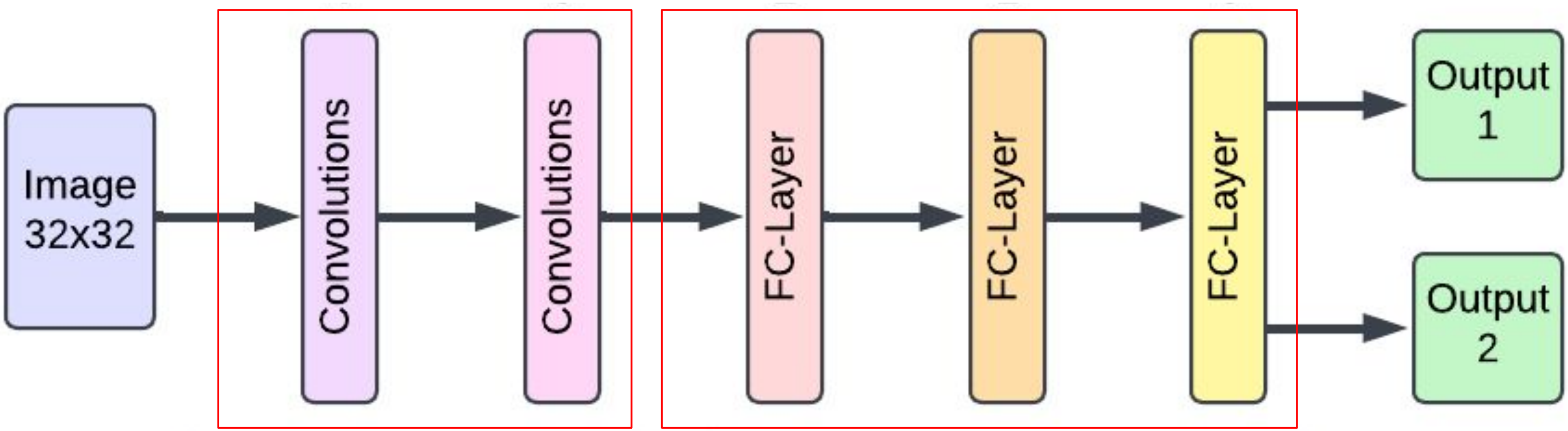


Performance of the Networks in Plasticity

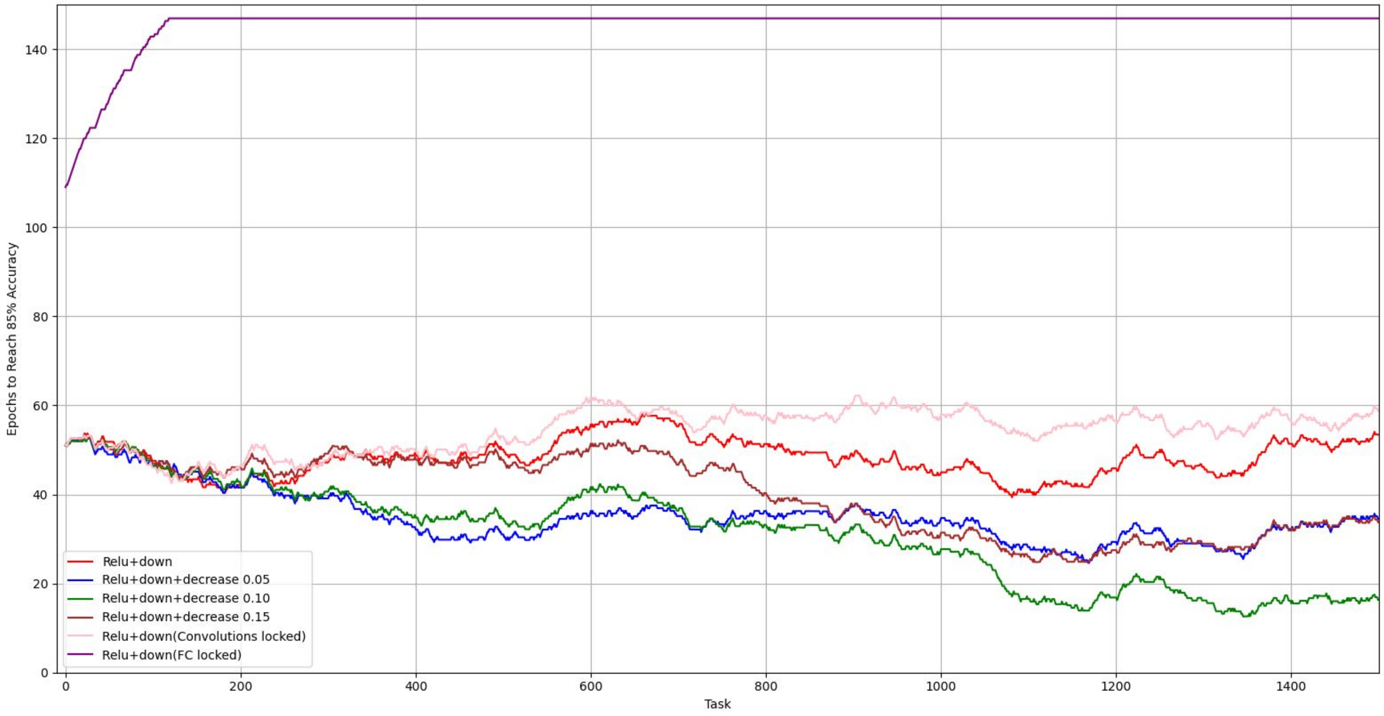


Continual Learning:
How Important is the Convolutional Part?

Locking different Parts of the Network



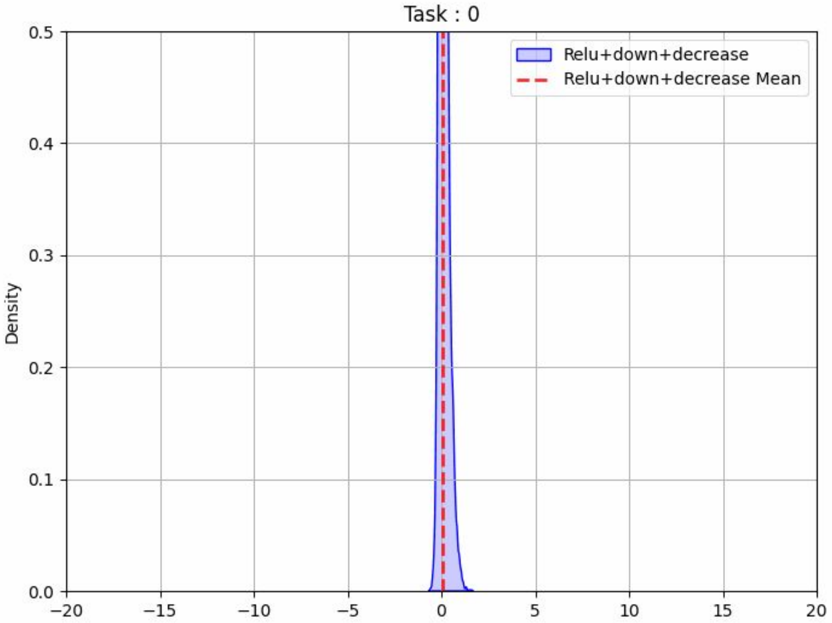
Performance of the Networks in Plasticity

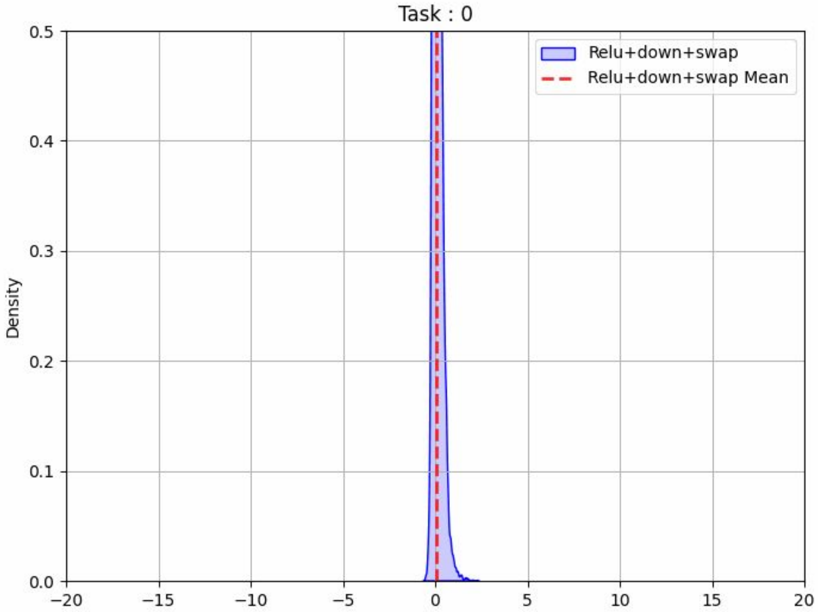


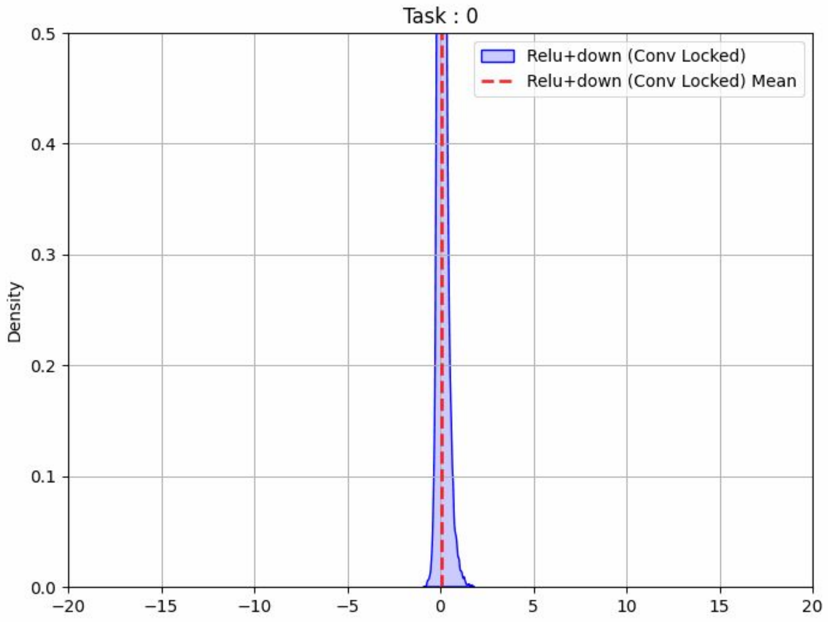
Continual Learning: Conclusion

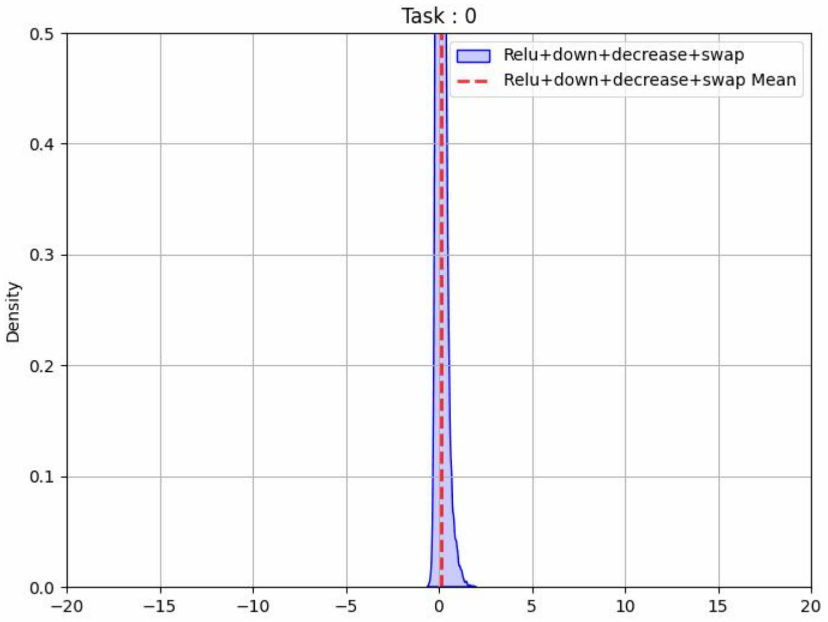
Conclusion

- Recap:
  - Implemented another way to evaluate Stability and Plasticity in Continual Learning
  - Found a way to maintain Plasticity via a simple Activation Function
  - Explored 2 ideas to have better Stability
    - Both of them did not improve Stability impactful
  - A decrease in BP based on Layers and Task leads to better Transfer Learning
- Future Work:
  - Exploring the Loss Landscapes of these different Approaches
  - Fixed Kernels(Systems Engineering) instead of Learned Convolutions
  - No Finetuning: Deriving the decrease in BP from the Data
- My Opinion
  - Decreasing Backpropagation could be a good Algorithm for Continual Learning
  - I think i will go back to the Standard way of Evaluating
- Questions?


Task : 0
Relu+down+decrease
Relu+down+decrease Mean
Density
0.0
0.1
0.2
0.3
0.4
0.5
−20
−15
−10
−5
0
5
10
15
20


Task : 0
Relu+down+swap
Relu+down+swap Mean
Density
0.0
0.1
0.2
0.3
0.4
0.5
−20
−15
−10
−5
0
5
10
15
20


Task : 0
Relu+down (Conv Locked)
Relu+down (Conv Locked) Mean
Density
0.5
0.4
0.3
0.2
0.1
0.0
−20
−15
−10
−5
0
5
10
15
20


Task : 0
Relu+down+decrease+swap
Relu+down+decrease+swap Mean
Density
0.5
0.4
0.3
0.2
0.1
0.0
−20
−15
−10
−5
0
5
10
15
20