University of Trieste
`Advanced Topics in Machine Learning` Course
by Prof. Fabio Anselmi
Academic Year 2023–2024

---

# On the Linearization of Convolutional Layers for FashionMNIST Dataset

Cristian Curaba Filippo Olivetti

Friday 29th December, 2023

## Contents

## 1 Introduction

These pages aim to report and make comments on the results obtained from the developed pipeline assigned by "Challenge 2" of the "Advance Topics in Machine Learning 2023/2024" course. The developed assignments are summarized in the following bullets:

- build a CNN architecture for classifying Mnist Fashion images;
- replace the `ReLU` function with the $\texttt{ReLU}_\alpha$ function, defined as follows:

  $$\texttt{ReLU}_\alpha(x) = \texttt{ReLU}(x) - (1-\alpha)\texttt{ReLU}(-x);$$

  where each layer has its own $\alpha$ parameter.
- consider a loss function $\tilde{\mathcal{L}}$ which penalizes $\alpha$ values;
- confront performance results with "fully" `ReLU` layers ($\alpha = 1$) and no `ReLU` layers ($\alpha = 0$);
- plot $\alpha$ values over training iteration and comment;
- comment on differences between linearized layers and fully `ReLU` ones, if possible.

## 2 Data preprocessing

We imported our training and test set of FashionMNIST from `torchvision` library. We fix the following:

- train data is given by 60000 images;
- test data is given by 10000 images;
- the *batch_size* is fixed to 100;
- number of classes is 10.

No transformations are performed.

## 3 Convolutional Neural Network Architecture

The chosen architecture is the following:

conv1: (1, 8, 5, 1, 2)
$\texttt{ReLU}_\alpha(x)$: torch.Size([100, 8, 28, 28])
conv2: (8, 16, 5, 1, 2)
$\texttt{ReLU}_\alpha(x)$: torch.Size([100, 16, 28, 28])
conv3: (16, 32, 3, 1, 1)
$\texttt{ReLU}_\alpha(x)$: torch.Size([100, 32, 28, 28])
fc1: torch.Size([100, 10])

where the convolutional tuple represents (in channels, out channels, kernel size, stride, padding) and in the torch size we have (batch size, number of channels, width, height).

So, we concatenated three convolutional layers with $\texttt{ReLU}_\alpha(x)$ activation and then we finished with a fully connected neural network for classifying.

We want to point out that, at the first attempt, we considered a more complex architecture where the batch norm[1] and max pool were added at each layer. We discarded this architecture because no performance differences resulted between considering the fully `ReLU` activation function and completely removing it from the layers. This is due to the non-linearity effects of batch norm and max pooling (hence, the architecture was not useful for our purpose).

## 4 Training

We define the loss function $\tilde{\mathcal{L}} = \texttt{Cross Entropy Error} + \lambda(|\alpha_1| + |\alpha_2| + |\alpha_3|)$ where $\alpha_1, \alpha_2$ and $\alpha_3$ (one for each of the 3 convolutional layer) represent the value $\alpha$ in the $\texttt{ReLU}_\alpha()$ activation function. They are initialized as follows: $\alpha_1 = \alpha_2 = \alpha_3 = 0.5$.

We used the Adam optimizer (with `learning_rate`=0.001) for updating weights and $\alpha_1, \alpha_2, \alpha_3$ values.

For the training loop, we fixed the following hyperparameters (if not specified differently):

- `number of epochs` = 3;
- `learning rate` = 0.001.

The model weights are randomly initialized while $\alpha_1 = \alpha_2 = \alpha_3 = 0.5$ as stated before.

We trained and tested the model for $\lambda$ in $\{0.1, 0.3, 0.5, 0.7, 0.9, 2, 5\}$ and then retrained with $\lambda$ in $\{0.5, 0.7\}$ but for `num_epoch`=12 to reach convergence for the $\alpha$ values.

## 5 Model performance

We will show the model accuracy for:

- CNN with `ReLU`;
- CNN without activation function;
- CNN with $\texttt{ReLU}_\alpha$ for $\lambda$ in $\{0.1, 0.5, 0.7, 2.0\}$ (for these cases we will show the tracking over iterations of the $\alpha_1, \alpha_2, \alpha_3$ values);
- CNN with $\texttt{ReLU}_\alpha$ over 12 epochs for $\lambda$ in $\{0.5, 0.7\}$ (for these cases we will show the tracking over iterations of the $\alpha_1, \alpha_2, \alpha_3$ values);

[1]reference: https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

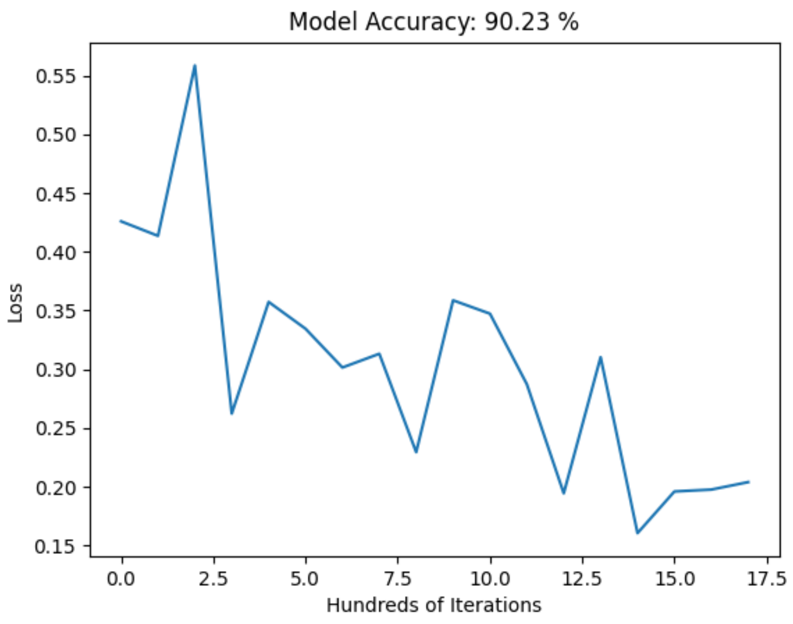

Figure 1: Convolutional Neural Networks with `ReLU`

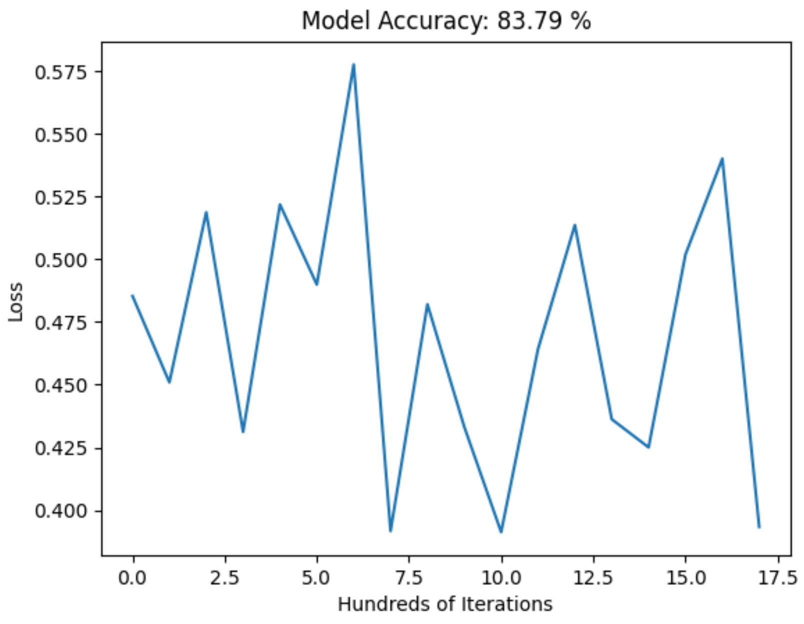

Figure 2: Convolutional Neural Networks without activation functions: the performance has decreased by 7.5% and loss values have high oscillations.

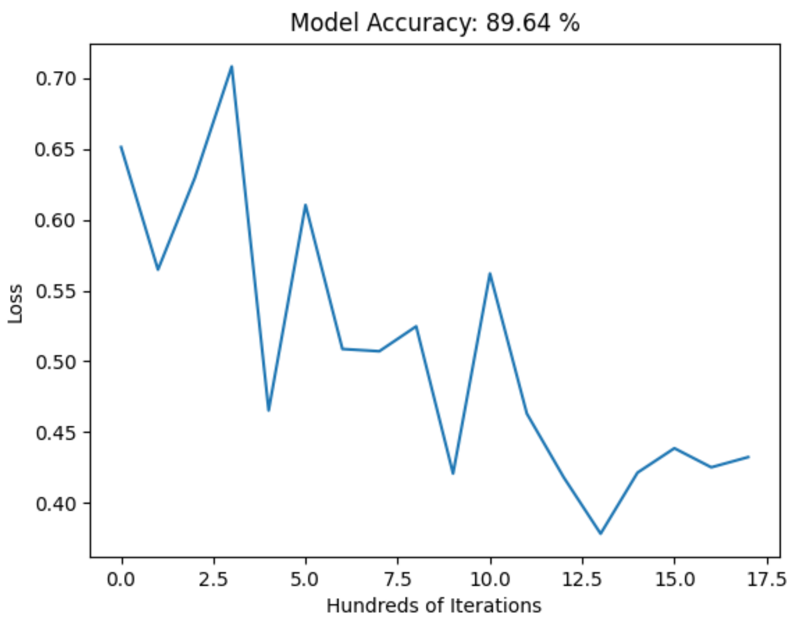

Figure 3: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.1$

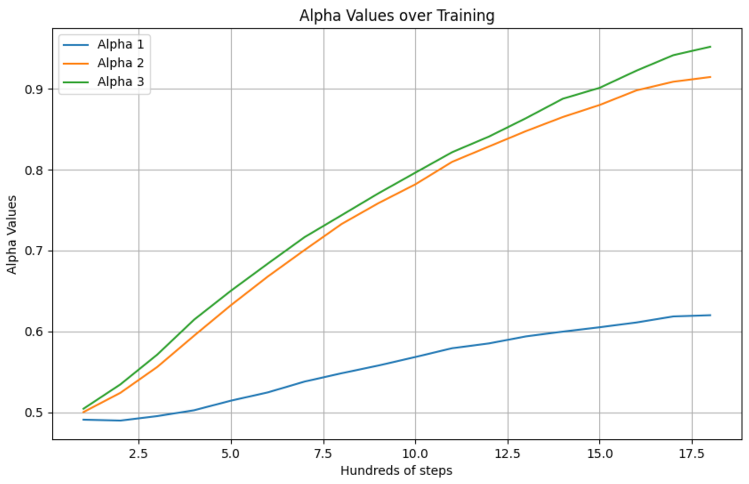

Figure 4: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.1$

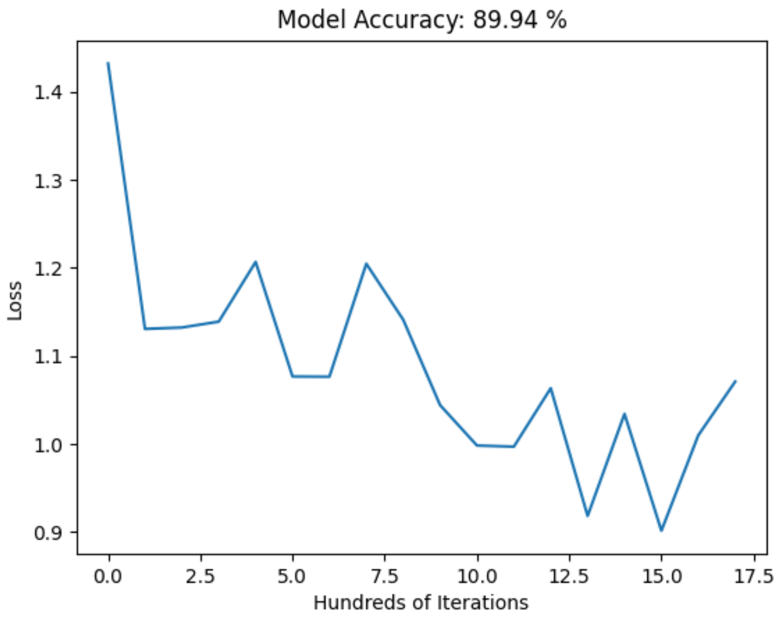

Figure 5: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.5$

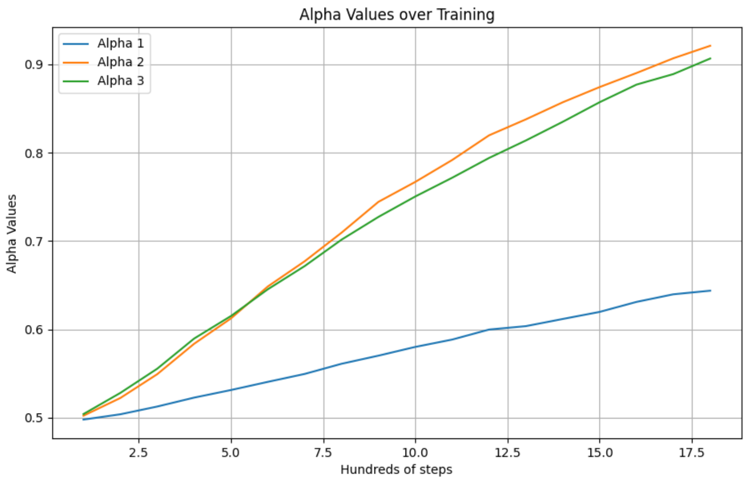

Figure 6: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.5$

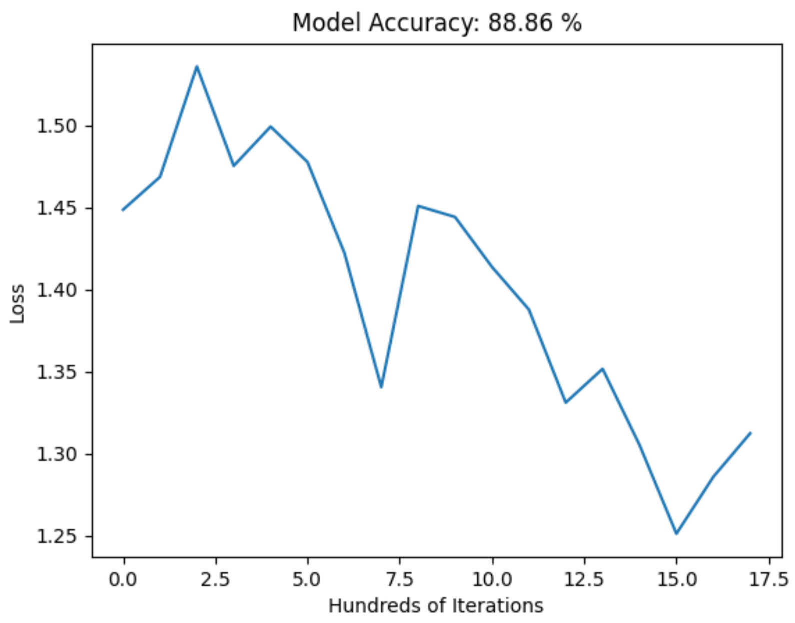

Figure 7: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.7$

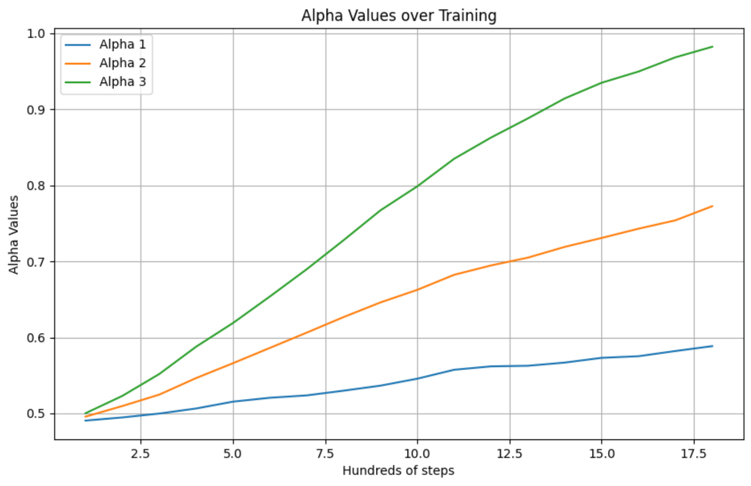

Figure 8: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.7$

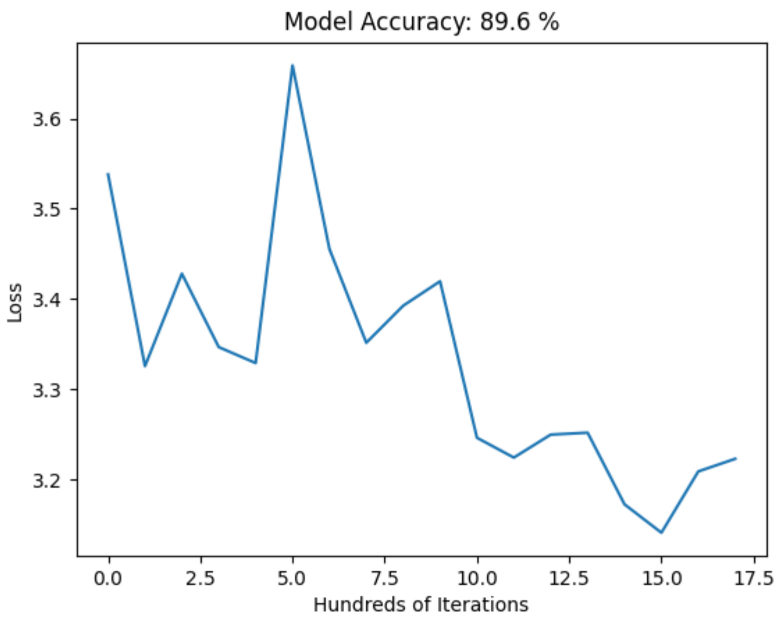

Figure 9: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 2.0$

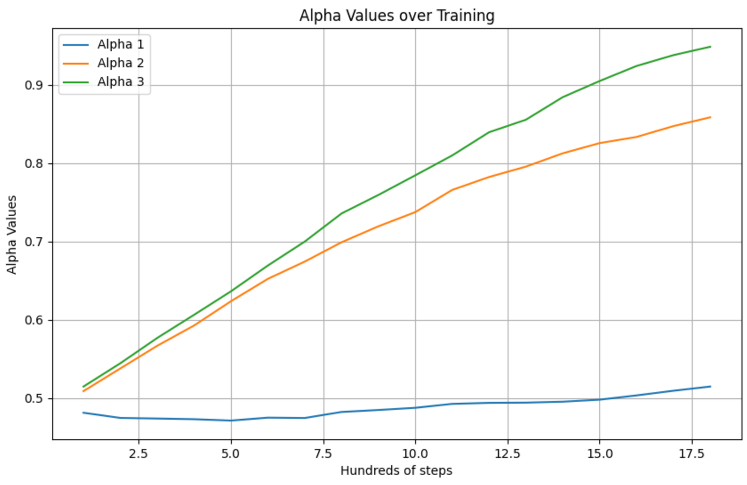

Figure 10: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 2.0$

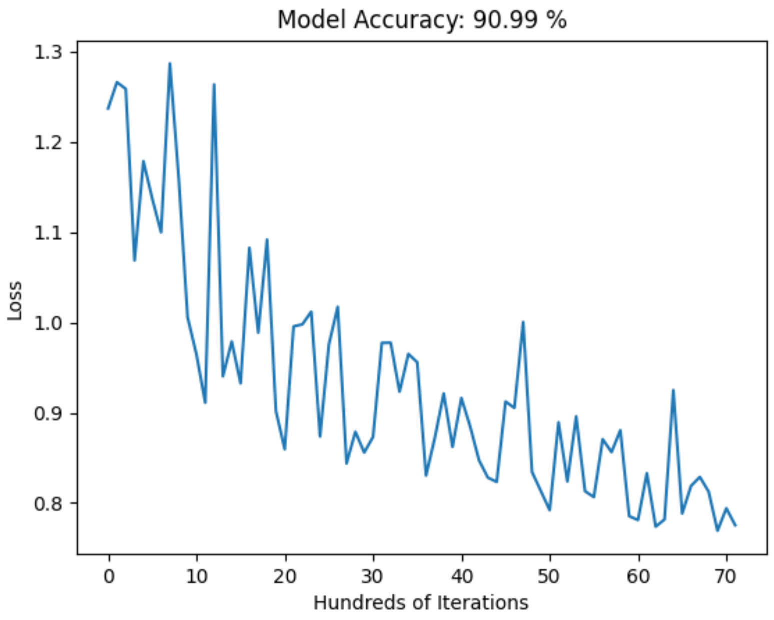

Figure 11: Convolutional Neural Networks with $\mathtt{ReLU}_\alpha$ and $\lambda = 0.5$

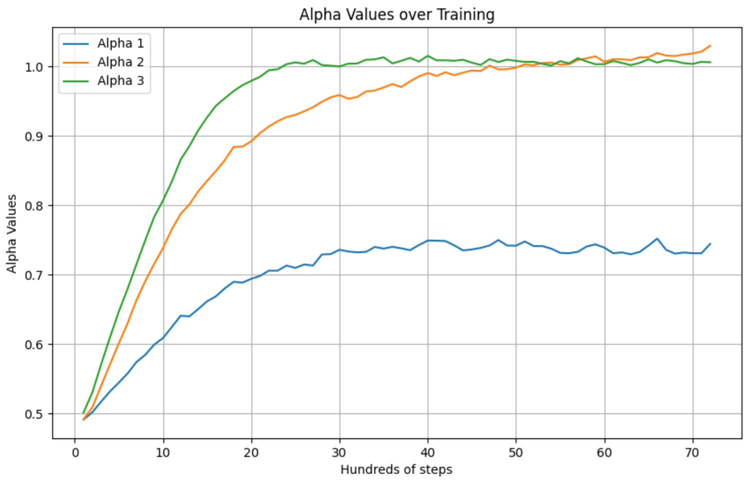

Figure 12: Convolutional Neural Networks with $\mathtt{ReLU}_\alpha$ and $\lambda = 0.5$

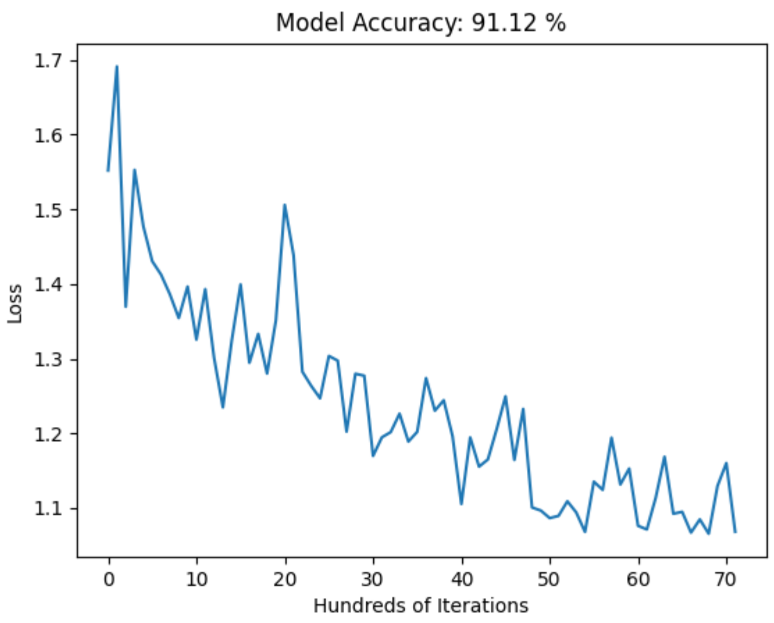

Figure 13: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.7$

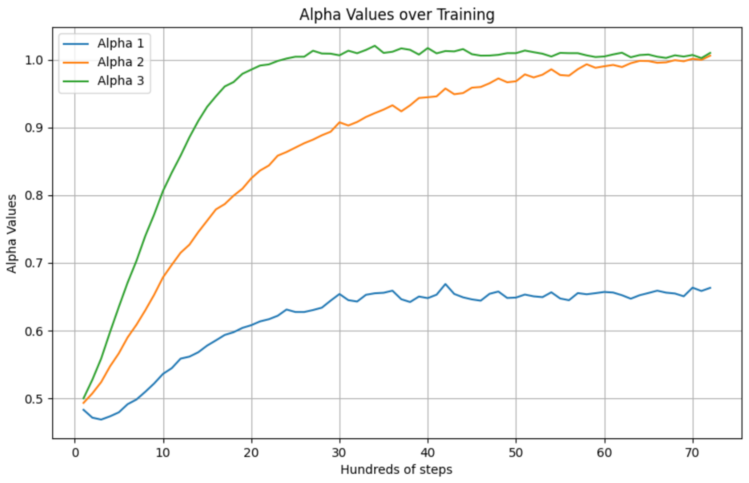

Figure 14: Convolutional Neural Networks with $\texttt{ReLU}_\alpha$ and $\lambda = 0.7$

## 6 Conclusion

We can notice that we achieve the same accuracy levels by considering the $\texttt{ReLU}_\alpha$ instead of `ReLU` activation function. Moreover, with all $\lambda$ values, the parameters $\alpha_2, \alpha_3$ rapidly jump to 1, while $\alpha_1$ tends to stabilize over a constant value in the interval $[0.5, 0.8]$ (as shown by the plots).

These results are interesting since we obtained a coherent behaviour of the $\alpha$ parameters independently from $\lambda$. In addition, it's surprising to see that $\alpha = 1$ is exactly an equilibrium (neither $\alpha = 0.9$ nor $\alpha = 1.1$) independently from the penalization parameter value. This tells us that `ReLU` is the perfect activation function for some layers, without the need to multiply it by a constant or consider a sharper non-linearity [2].

The results also support the idea that non-linearities are fundamental for nice model architectures because, even when penalized, the $\alpha$ values tend to increase (or at least one of them). We also pondered the issue that $\alpha_1$ remains far below 1. A possible explanation of that fact is that at the first stage of the architecture adding non-linearities to the NN is not as critical as it is in the following layers which are usually capable of catching complex and highly non-linear patterns in the input image.

We also tried to interpret the models by feeding input images and plotting the transformations after each layer. We noticed that, as already observed in the literature, the non-linear layers try to highlight borders and transform the image into a human-unrecognizable image.

Instead, the linear layers seem to "blur" the image (probably the classification becomes more robust to small translations a.k.a layer transformations are "small translation equivariant") and the final image is still human-recognizable. We reported three input tests (boot, shirt and trousers) each with three different models (`ReLU`, only linear, $\texttt{ReLU}_\alpha$ with $\lambda = 0.5$ trained in 12 epochs).

[2] for example, with $\alpha = 2$ we get that $\texttt{ReLU}_2(x) = abs(x)$.

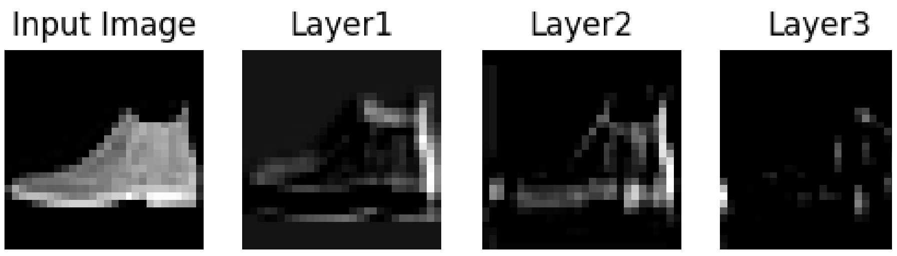

Figure 15: Correctly classified input image through the 3 layers of the `ReLU` CNN model.

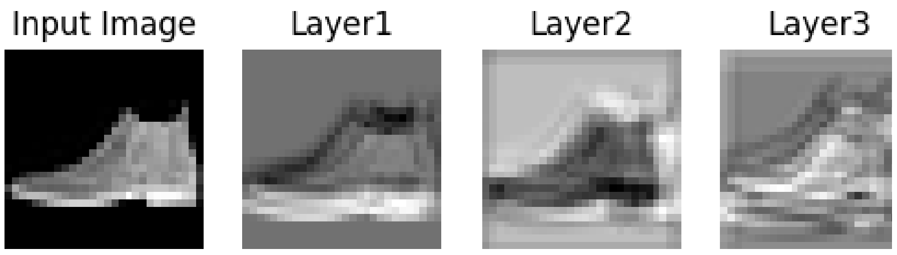

Figure 16: Correctly classified input image through the 3 layers of the CNN model without activation functions.

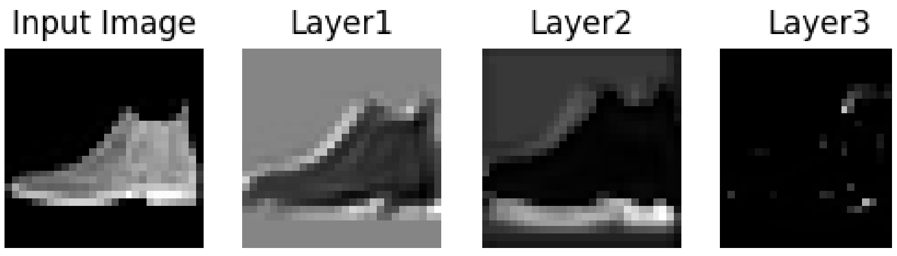

Figure 17: Correctly classified input image through the 3 layers of the CNN model with $\texttt{ReLU}_\alpha$, $\lambda = 0.5$ and 12 epochs.

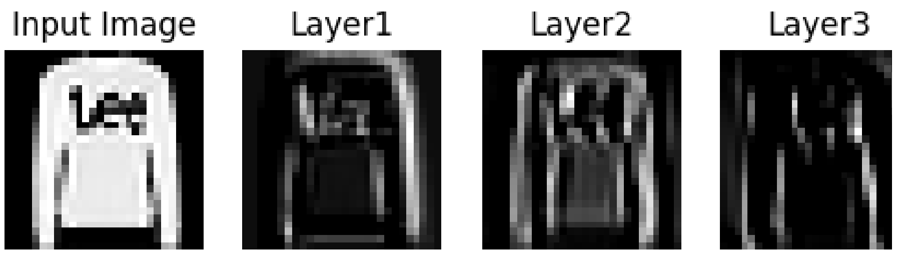

Figure 18: Correctly classified input image through the 3 layers of the `ReLU` CNN model.

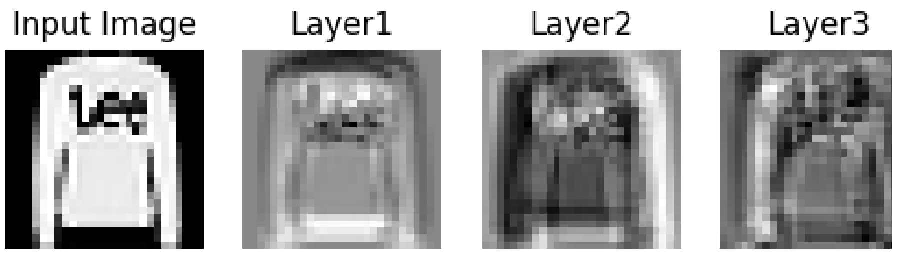

Figure 19: Correctly classified input image through the 3 layers of the CNN model without activation functions.

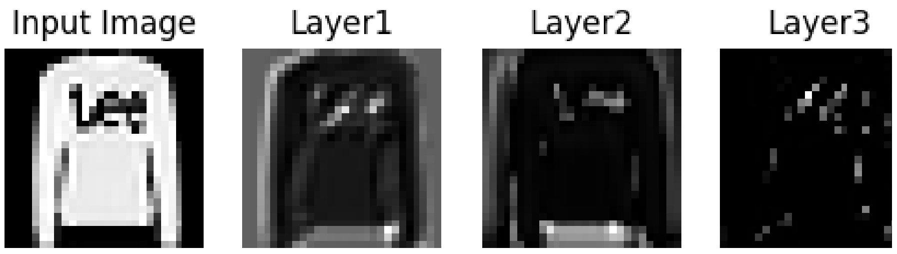

Figure 20: Correctly classified input image through the 3 layers of the CNN model with $\texttt{ReLU}_\alpha$, $\lambda = 0.5$ and 12 epochs.

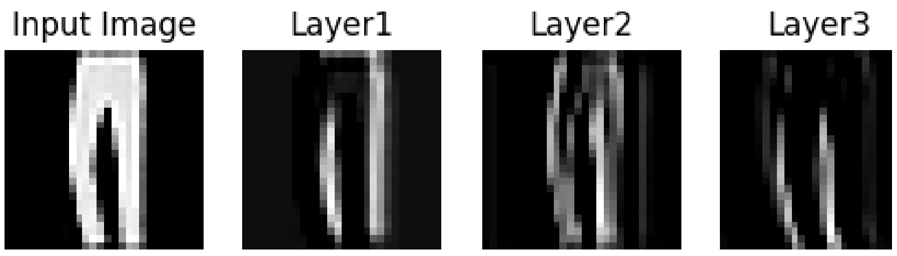

Figure 21: Correctly classified input image through the 3 layers of the `ReLU` CNN model.

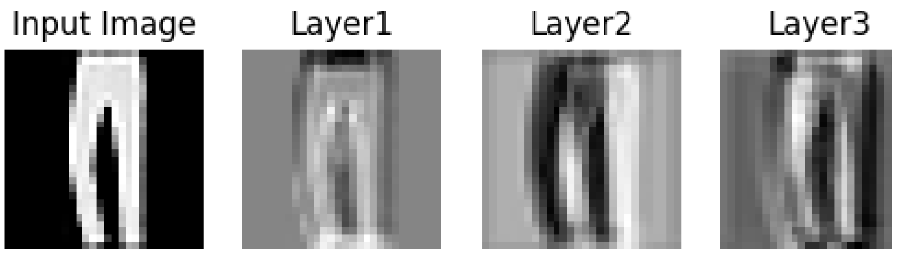

Figure 22: Correctly classified input image through the 3 layers of the CNN model without activation functions.

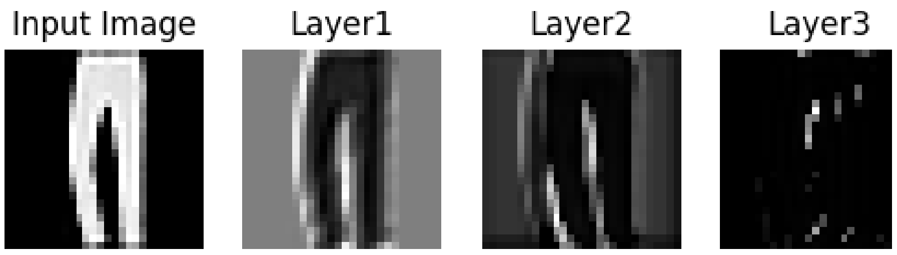

Figure 23: Correctly classified input image through the 3 layers of the CNN model with $\texttt{ReLU}_{\alpha}$, $\lambda = 0.5$ and 12 epochs.