A Note On The Stability Of The Focal Loss

Anonymous authors
Paper under double-blind review

Abstract

The Focal loss is a widely deployed loss function that is used to train various types of deep learning models. This loss function is a modification of the cross-entropy loss designed to mitigate the effect of class imbalance in dense object detection tasks by downweighing easy, well-classified examples. In doing so, more focus is placed on hard, wrongly-classified examples by preventing the gradients from being dominated by examples from which the model can easily predict the correct class. This downweighing is achieved by scaling the cross-entropy loss with a term that depends on a focusing parameter γ. In this paper, we highlight an unaddressed instability of the Focal loss that arises when this focusing parameter is set to a value between 0 and 1. We present the theoretical foundation behind this instability, show that it is numerically identifiable, and demonstrate it in a binary classification and segmentation task on the MNIST dataset. Additionally, we propose a straightforward modification to the original Focal loss to ensure stability whenever these unstable focusing parameter values are used.

1 Introduction

The Focal loss is a broadly used loss function for one-stage detectors, medical imaging, segmentation, and pose estimation (Terven et al., 2023). This function modifies the distribution-based cross-entropy loss by introducing a focusing parameter (γ) that downweighs the penalty applied to "easy" examples (Lin et al., 2017). The downweighing of losses prevents the gradients from being dominated by easy examples, allowing for an increased focus on difficult examples (Lin et al., 2017). This is especially useful when the training data comprises a high proportion of the background (or another) class.

The selection of parameter γ, and with this, the extent to which these easy examples are downweighted, should be done via cross-validation (Lin et al., 2017). The original Focal loss reported that using a γ of 2 led to the best results in their experiments (Lin et al., 2017). There is, however, a limit to what γ values can be used. Selecting γ values much larger than 2 has been found to result in gradients close to 0 for relatively low model outputs, causing training to fail (Mukhoti et al., 2020). This paper will shed light on the other end of the spectrum, showing that the Focal loss gradients can become unstable whenever γ is too small. More specifically, we address an instability of the Focal loss that arises whenever a γ is set to a value between 0 and 1. These γ values can cause the Focal loss derivative to become undefined and destabilize model training. The singularity arises whenever the Focal loss, in combination with a γ on the open unit interval, is used to learn a task for which a model can predict a correct class label with high confidence. We will demonstrate that the instability of the Focal loss is not only a theoretical problem by demonstrating that a simple convolutional neural network (CNN) and a 2D U-net can return undefined loss values during training whenever γ values on the open unit interval are used. Henceforth, we will refer to the γ values on the open unit interval as unstable γ values.

The original Focal loss paper (Lin et al., 2017) did not address the limitation we highlight in this paper, and contains experimental training results generated with unstable γ values. Their results indicate that training with these unstable γ values does not always cause instabilities. However, we show that under certain conditions, the singularity can be encountered.

We address the Focal loss instability by modifying the original Focal loss with a smoothing constant that eliminates the root of the singularity. By adopting the modification, the singularity can be removed without altering the original behavior of the Focal loss. Our modification, therefore, does not hamper already existing methods that use the Focal loss with unstable γ values, such as the Unified Focal loss (Yeung et al., 2022).

2 Methods

This section will review the definition of the Focal loss and its derivative to explain the origin of its instability. Following the analytical derivation of the instability, we found that computing the gradients of the Focal loss with the original Focal loss function while using unstable γ values can result in undefined gradients. Additionally, to show that this instability is not only a theoretical obstacle, we will demonstrate that under certain conditions, the instability can be induced in a binary classification and 2D segmentation task. Lastly, we show how to modify the original Focal loss to eliminate the instability whenever γ values between 0 and 1 are used.

2.1 Cross-entropy and the Focal Loss

The Focal Loss was introduced to address class imbalance by reducing the effect of easily classifiable examples, thereby placing more emphasis on harder, misclassified ones (Lin et al., 2017). This is achieved by modulating the standard cross-entropy loss in Equation (1) with a scaling factor based on the prediction error and a focusing parameter γ. This modulating factor ensures that the smaller the prediction error becomes, the more the cross-entropy is downscaled. In other words, predictions that are closer to the correct label (easy examples) are downscaled by the focusing parameter γ. Note that whenever a γ of 0 is used, the Focal loss simplifies to the cross-entropy loss. For simplicity, without loss of generality, we will simplify the cross-entropy loss function to the binary cross-entropy loss and reformulate the equations to a foreground ($\mathcal{L}_{fg}$) and background loss ($\mathcal{L}_{bg}$). Nevertheless, the derivations shown below also hold for the multiclass cross-entropy. For consistency, we make use of the same notation for the ground truth (y) and model output (p) as was used in the original Focal loss paper (Lin et al., 2017). While the original Focal Loss paper reformulates the loss as a foreground loss for notational convenience, this paper explicitly highlights both the foreground and background components of the Focal Loss.

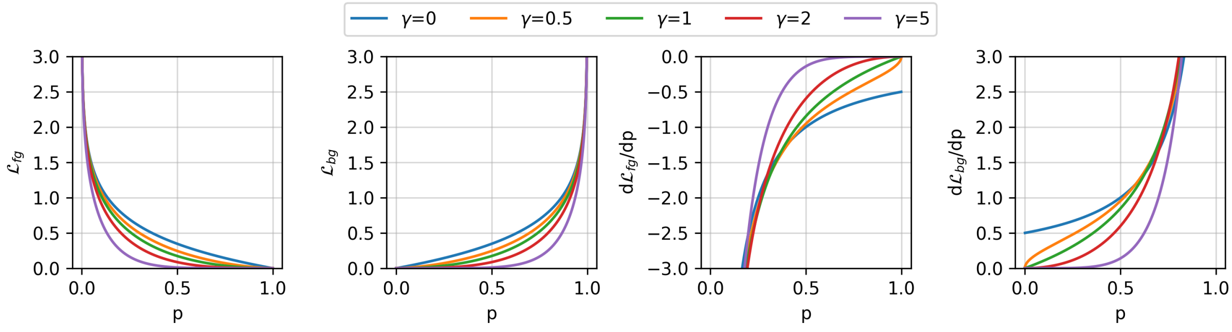
$$\mathcal{L}_{\text{CE}} = -\underbrace{y\log(p)}_{\mathcal{L}_{fg}} - \underbrace{(1-y)\log(1-p)}_{\mathcal{L}_{bg}} \tag{1}$$

$$\mathcal{L}_{\text{F}} = \underbrace{-\alpha_t y\,(1-p)^{\gamma}\log(p)}_{\mathcal{L}_{fg}} - \underbrace{(1-\alpha_t)(1-y)\,p^{\gamma}\log(1-p)}_{\mathcal{L}_{bg}} \tag{2}$$

The Focal loss, as defined in Equation (2), downscales both the foreground and background loss equally with the focusing parameter γ. As shown in Equation 2, the Focal loss also includes a parameter α_t that is used to scale the contribution of the foreground and the background loss relative to each other.

2.2 Derivative of the Focal Loss

Figure 1a and Figure 1b shows the foreground and background component of the Focal loss for different model outputs p when changing the value for the focusing parameter γ. These plots show that an increase in γ will cause the Focal loss to show near-zero loss values for model outputs close to the ground truth label, consequently lowering their associated gradients. As we previously decomposed the Focal loss into a foreground and background loss, we can define the Focal loss derivative as the sum of the foreground and background components, as shown in Equation (3). These two components are defined in Equations (4) and (5), and are displayed in Figure 1c and Figure 1d. A more detailed derivation of these equations is included in Appendix A.1.



(a) (b) (c) (d)

Figure 1: Foreground (a) and background (b) component of the Focal loss and their associated foreground (c) and background (d) derivative. The losses and derivatives were calculated with different γ values and a fixed α_t of 0.5.

$$\frac{d\mathcal{L}_F}{dp} = \frac{d\mathcal{L}_{fg}}{dp} + \frac{d\mathcal{L}_{bg}}{dp} \tag{3}$$

$$\frac{d\mathcal{L}_{fg}}{dp}|_{y=1} = \alpha_t \left(\gamma (1-p)^{\gamma-1} \log(p) - \frac{(1-p)^{\gamma}}{p} \right) \tag{4}$$

$$\frac{d\mathcal{L}_{bg}}{dp}|_{y=0} = -(1-\alpha_t) \left(\gamma\, p^{\gamma-1} \log(1-p) - \frac{p^{\gamma}}{1-p} \right) \tag{5}$$

2.3 Focal Loss Instability

In the derivative of the Focal loss, a $(\gamma - 1)$ exponent is introduced in both the foreground and background loss. Whenever $0 < \gamma < 1$, this $(\gamma - 1)$ term becomes negative, and the model output is raised to the power of a negative number, leading to a fraction with the model output in the denominator. An example of this is illustrated in Equations (6) and (7), showing the derivatives of the foreground and background loss for $\gamma = 0.5$.

$$\frac{d\mathcal{L}_{fg}}{dp}|_{y=1,\gamma=0.5} = \alpha_t \left(\frac{0.5}{\sqrt{1-p}} \log(p) - \frac{\sqrt{1-p}}{p} \right) \tag{6}$$

$$\frac{d\mathcal{L}_{bg}}{dp}|_{y=0,\gamma=0.5} = -(1-\alpha_t) \left(\frac{0.5}{\sqrt{p}} \log(1-p) - \frac{\sqrt{p}}{1-p} \right) \tag{7}$$

The limits as p approaches 1 for the foreground and 0 for the background are presented in Equation (8) and (9). These equations show that in these limits, the fraction introduced in the derivative leads to a division by 0, creating a singularity that causes training instability. Although in this example γ was set to 0.5, this holds for all γ values between 0 and 1, as all these values introduce a fraction in the derivative with the model output in the denominator.

$$\lim_{\hat{y} \to 1} \frac{d\mathcal{L}_{fg}}{dp}|_{y=1,\gamma=0.5} = \alpha_t \left(\frac{0}{\sqrt{0}} - 0 \right) \tag{8}$$

$$\lim_{\hat{y} \to 0} \frac{d\mathcal{L}_{bg}}{dp}|_{y=0,\gamma=0.5} = -(1-\alpha_t) \left(\frac{0}{\sqrt{0}} - 0 \right) \tag{9}$$

Consider training a binary classification model with the Focal loss and a γ value of 0.5, which should distinguish a foreground class from a background class. When the foreground is easily separated from the background class, the model will assign high values to the model outputs representing the correct class. Whenever these model outputs approach the ground truth value ($y \approx p$), the derivative will become undefined due to the division by 0, consequently triggering the instability. When faced with a more challenging

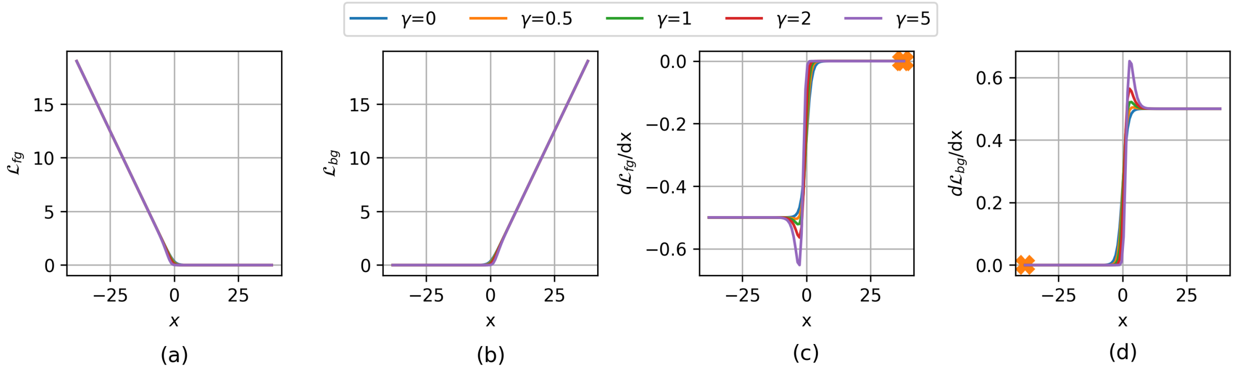


Figure 2: Computed values for the foreground (a) and background (b) components of the Focal loss in combination with the foreground (c) and background (d) gradients. These losses and gradients were computed with model outputs that were not yet processed by a the sigmoid (x) and ranged between [-38,38]. Both the losses and gradients were computed with the *torchvision.ops.sigmoid_focal_loss* function with an α_t of 0.5, and different values for γ. The orange cross indicates the value for x that causes "NaN" values when computing the loss.

task, the classification model is unlikely to produce model outputs that closely approximate the ground truth, thereby preventing instability from being triggered. Since most deep learning tasks are complex, this is presumably why the instability is not always an issue and why it has not yet been addressed in the literature.

One important note to consider when discussing the stability of the Focal loss is that whenever the opposite of the ground truth is predicted by the model, a log(0) is introduced in the equation of the Focal loss, which also causes instabilities. Note that the instability that this paper addresses occurs whenever the model produces output values that are nearly equal to the ground truth. In other words, the log(0) instability occurs when the prediction error becomes extremely large, whereas the instability that we address occurs whenever the prediction error becomes near-zero. Since machine learning models are optimized to minimize the prediction error, models are optimized towards a state where this instability will eventually occur, emphasizing the importance of resolving this instability.

2.3.1 Numerical Gradient Computation

Here we show that the instability also emerges when computing the gradients with the original Focal loss function (torchvision.ops.sigmoid_ focal_loss) used in the seminal work (Lin et al., 2017). This implementation of the Focal loss applies a sigmoid activation function to the model outputs as shown in Equation (10), in which the model output before applying the sigmoid is defined as x. After applying the sigmoid, the Focal loss is calculated using Equation (2). Figure 2a and Figure 2b show the foreground and background components of the Focal loss when computing them with the original Focal loss function.

$$p = \sigma(x) = \frac{1}{1+e^{-x}} \tag{10}$$

Similar to Figure 1, Figure 2 shows the foreground and background loss in combination with their associated gradients. Figure 1 shows the Focal loss for model outputs that are processed by a sigmoid, and Figure 2 shows the Focal loss for unprocessed model outputs. These unprocessed output values are not confined to a range of [0,1] but can span from $[-\infty,\infty]$. Figure 2c and Figure 2d present the computed Focal loss gradients for the foreground and background components of the Focal loss. Similar gradients are presented in the original Focal loss paper, but instead of using an α_t of 0.5, they computed the gradients with an α_t of 1. The orange 'X' markers in Figure 2 indicate the output value that causes the gradient to become undefined, consequently returning an "NaN". This shows that the instability can also occur when computing the loss with the loss function published by the original Focal loss paper.

2.4 Stabilized Focal Loss

As described in the previous section, the instability of the Focal loss arises as a result of a division by 0 in its derivative. One commonly used method to prevent a division by 0 in a loss function is the introduction of a smoothing constant ϵ in the denominator of the loss. This method is also applied to the well-known Dice loss (Milletari et al., 2016; Sudre et al., 2017). We propose a modification of the original Focal loss that leads to the introduction of a smoothing constant in the denominator of the derivative when unstable γ values are used. This is slightly different from what is done to stabilize the Dice loss, in which the division by zero is prevented in the loss itself. Instead, we modify the original loss with a parameter ϵ, so that the ϵ term is placed in the denominator of the derivative, preventing division by zero when computing the gradient with unstable γ values.

The modified Focal loss and the derivatives for its foreground and background components are defined as shown in Equations (11), (12), and (13). The modification ensures that when a γ value between 0 and 1 is used, the derivative will have a constant term in the denominator that is independent of the model output, preventing the division by 0 when the prediction approaches 0, eliminating the Focal loss's instability. Note that smaller γ values will cause the denominator to approach 0 more quickly for model outputs close to the ground truth, compared to when a larger value of γ is used. This means that a larger ϵ is required to ensure stability whenever a smaller γ value is used. We ran the experiments in this paper with a value of ϵ equal to $1e-3$, as it stabilized model training whenever a γ as small as 0.1 was used.

$$\mathcal{L}_{\text{Fa}} = \underbrace{-\alpha_t y\,(1-p+\epsilon)^{\gamma}\log(p)}_{\mathcal{L}_{fg}} - \underbrace{(1-\alpha_t)(1-y)\,(p+\epsilon)_i^{\gamma}\log(1-p)}_{\mathcal{L}_{bg}} \tag{11}$$

$$\frac{d\mathcal{L}_{fg}}{dp}|_{y=1} = \alpha_t\left(\gamma(1-p+\epsilon)^{\gamma-1}\log(p) - \frac{(1-p+\epsilon)^{\gamma}}{p}\right) \tag{12}$$

$$\frac{d\mathcal{L}_{bg}}{dp}|_{y=0} = -(1-\alpha_t)\left(\gamma\,(p+\epsilon)^{\gamma-1}\log(1-p) - \frac{(p+\epsilon)^{\gamma}}{1-p}\right) \tag{13}$$

When revisiting the example in which γ is equal to 0.5, the derivatives for the foreground and background loss become equal to Equation (14) and (15). The limits for these equations, as shown in Equation (16) and (17), show that whenever the model output p approaches 1 in the foreground loss and 0 in the background loss, the division by zero is prevented by the smoothing constant ϵ. The implementation details of the modified version of the Focal loss can be found in Appendix A.3.

$$\frac{d\mathcal{L}_{fg}}{dp}|_{y=1,\gamma=0.5} = \alpha_t\left(\frac{0.5}{\sqrt{1-p+\epsilon}}\log(p) - \frac{\sqrt{1-p+\epsilon}}{p}\right) \tag{14}$$

$$\frac{d\mathcal{L}_{bg}}{dp}|_{y=0,\gamma=0.5} = -(1-\alpha_t)\left(\frac{0.5}{\sqrt{p+\epsilon}}\log(1-p) - \frac{\sqrt{p+\epsilon}}{1-p}\right) \tag{15}$$

$$\lim_{\hat{y}\to 1}\frac{d\mathcal{L}_{fg}}{dp}|_{y=1,\gamma=0.5} = \alpha_t\left(\frac{0}{\sqrt{\epsilon}} - \frac{\sqrt{\epsilon}}{1}\right) = -\alpha_t\sqrt{\epsilon} \tag{16}$$

$$\lim_{\hat{y}\to 0}\frac{d\mathcal{L}_{bg}}{dp}|_{y=0,\gamma=0.5} = -(1-\alpha_t)\left(\frac{0}{\sqrt{\epsilon}} - \frac{\sqrt{\epsilon}}{1}\right) = (1-\alpha_t)\sqrt{\epsilon} \tag{17}$$

2.5 Experiments

To demonstrate the instability of the Focal loss, we conducted two experiments. In the first experiment, we tested whether the instability can occur when training a basic CNN (shown in Appendix A.4) to perform a binary classification task on the MNIST dataset (Deng, 2012). In the second experiment, we tested whether the instability can be detected in a segmentation task when training for a 2D U-Net (Ronneberger et al., 2015) (shown in A.6). This subsection will describe how these two experiments were set up.

Table 1: The number of samples in each class of the MNIST dataset (Deng, 2012) when using different class distributions. The class distribution was determined by a binarization threshold ranging between 0 and 8. The threshold value indicates the cut-off value for the classes belonging to class A or class B. All values larger than the threshold belong to class B, and all classes below or equal to the threshold to class A.

Threshold	Samples Class A	Samples Class B	Class A/B ratio
0	5.923	54.077	0.11
1	12.665	47.335	0.27
2	18.623	41.377	0.45
3	24.754	35.246	0.70
4	30.596	29.404	1.04
5	36.017	23.983	1.50
6	41.935	18.065	2.32
7	48.200	11.800	4.08
8	54.051	5.949	9.09

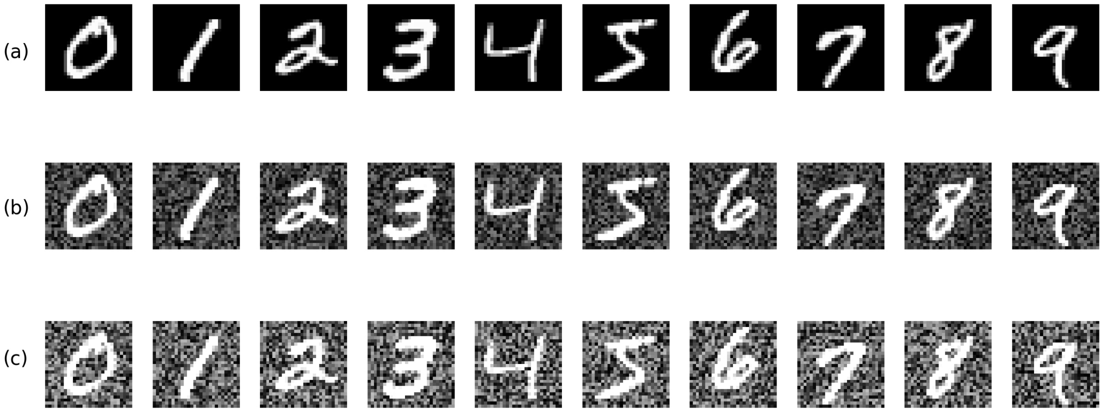


Figure 3: Example of the input images from the MNIST dataset (Deng, 2012) with classes 0 to 9. *(a)* shows the original images, *(b)* shows the images with "Medium Noise" added, and *(c)* shows the images with "High Noise" added. The "Medium Noise" was generated by multiplying uniformly sampled noise between 0 and 1 by 0.5, while the "High Noise" was generated by multiplying this noise by 0.75.

2.5.1 Binary Classification

In the first experiment, we divided the MNIST dataset (Deng, 2012) into two classes, where a threshold determined which numbers belonged to which class. The goal of the CNN was to learn to distinguish between the two classes by training for 100 epochs. As we are interested in the stability of the Focal loss during training, we did not focus on model performance, but rather on whether the model could complete all 100 epochs without encountering any instabilities. We tested this for both stable (0,1,2,3,4,5) and unstable γ values ($0 < \gamma < 1$).

To transform the multiclass MNIST dataset into a dataset that can be used for a binary classification task, we applied a threshold to the MNIST class labels, restructuring the dataset into a dataset with two classes: a foreground (A) and background (B) class. The number of samples for each class of the MNIST training dataset is approximately the same (Hamidi & Borji, 2010), so by changing this threshold, the foreground-background class distribution could be changed incrementally. Assessing the effect of changing this threshold provides insight into whether imbalance in classes influences training stability whenever unstable γ values are used. For example, a threshold of 4 means that the MNIST numbers with classes 0-4 belong to class A and the classes 5-9 belong to class B. Increasing or decreasing this threshold would mean an increase in class imbalance. By changing the class distribution, we aimed to introduce a bias to the majority class, leading to confident predictions of the accurate label and, consequently, the expression of instability.

The initial part of the experiment determined whether simplifying the classification task influenced the instability. The second part of the experiment repeated the initial experiment, but trained the CNN with random noise added to the input images to evaluate whether an increased difficulty of the classification task mitigates the unstable behavior of the Focal loss. In this experiment, we added noise to the input images by sampling random values from a uniform distribution over [0, 1). The noise was multiplied by 0.5 and 0.75 to create what we refer to as "Medium Noise" and "High Noise," respectively. This randomly sampled noise was then added to the pixels of the input images. Examples of input images with and without added noise are shown in Figure 3.

2.5.2 2D Segmentation

In the second experiment, we tested whether we could induce the Focal loss instability when training a segmentation model to segment the numbers in the MNIST dataset using a 2D U-net. However, since the MNIST dataset is a classification dataset, it does not include segmentation masks. We therefore applied a threshold of 0.5 to the input images, setting all pixels that exceeded this value to the foreground of the segmentation mask, and setting all pixels below this value to the background class. Similar to the binary classification task, we repeated the experiment after adding noise to the input data, testing whether increasing the task's difficulty influenced training stability. The noise was added after creating the segmentation masks to maintain a consistent segmentation mask across experiments. After preprocessing of the data, the 2D U-net was trained with the Focal loss using a γ and α_t of 0.5 for 1000 epochs.

3 Results

In this section, we show the unstable behavior of the Focal loss when training a simplistic CNN and a 2D U-Net with γ values between 0 and 1. We present results when using the original Focal loss and show how training stability changes when training with our modified version of the Focal loss.

3.0.1 Binary Classification

For the binary classification task, we trained the CNN for 100 epochs with γ values ranging from 0 to 5 using the different class A/B ratios as shown in Table 1. Whenever a "NaN" was encountered during training, training was stopped, otherwise, training would continue until all 100 epochs were completed. The results for these experiments are presented in Figure 4, where the number of completed epochs is shown for all γ and A/B class ratios. The left plot in Figure 4 shows the training results when training with γ values of either 0 or larger than 1. This figure shows that for these γ values, all 100 epochs were completed, reporting no instabilities.

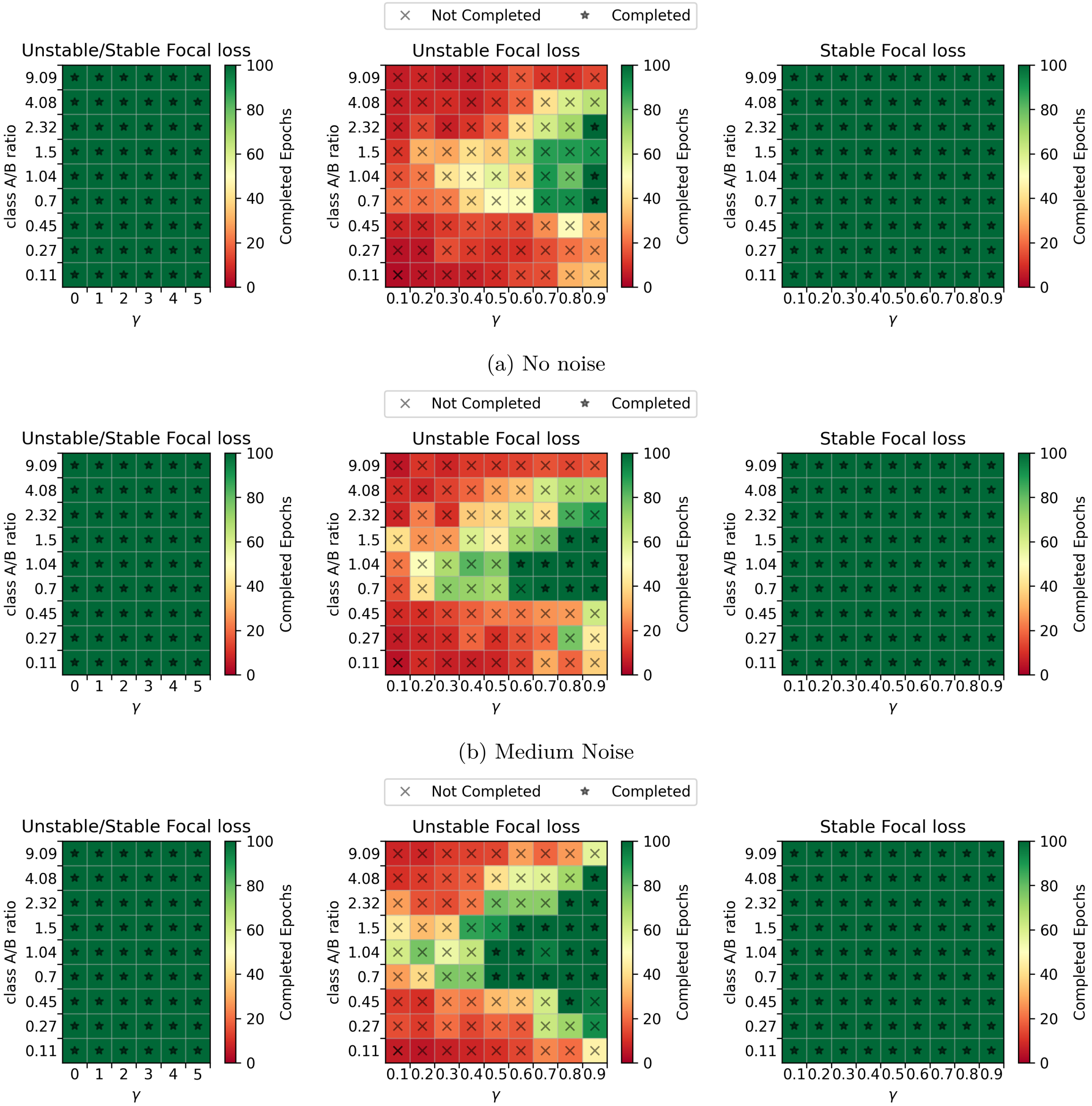
The figures in the middle column show the results when training with γ values ranging from 0.1 to 0.9 with increments of 0.1. From this figure, we see that using γ values between 0 and 1 causes instability in almost all cases. These instabilities quickly arise, especially for smaller γ values and unbalanced datasets. However, when using a γ value of 0.9, all 100 epochs were still completed whenever a balanced class distribution was used. It is, however, not unlikely that whenever these models were trained for more than 100 epochs, the instability would still be found in a later epoch.

Figure 4b and 4c show that adding noise has a mitigating effect on how quickly the instabilities are found. These figures show that larger amounts of noise lead to more γ values showing stable behavior. Additionally, adding noise allows for a broader range of class distributions to show stable training results whenever larger γ values are used.

After stabilizing the Focal loss, as proposed, all experiments were repeated, the results of which are shown in the last column of Fig. 4. These results show that the modification of the Focal loss successfully eliminated the instability, as no more instabilities are reported in any of the experiments.

3.0.2 2D Segmentation

The results of training the 2D U-net are shown in Fig. 5. This figure reports the computed Focal loss for each epoch, but halted training whenever a "NaN" was encountered or all 1000 epochs were completed. If no noise was added to the input data, model instability was quickly detected. Adding some noise to the



(a) No noise

(b) Medium Noise

(c) High Noise

Figure 4: *(a)*: Experiment results showing the number of completed epochs using different γ values. The left plot shows model results when using γ values of 0,1,2,3,4 and 5. The middle and left plots show the number of completed epochs whenever the γ values of 0.1 to 0.9 with increments of 0.1 are used, where the middle plot shows the results when training with the original Focal loss, and the last plot shows the results when using the stabilized Focal loss. *(b-c)*: Experiment results when the initial experiment is repeated with additional noise added to the input images, increasing the difficulty of the classification task.

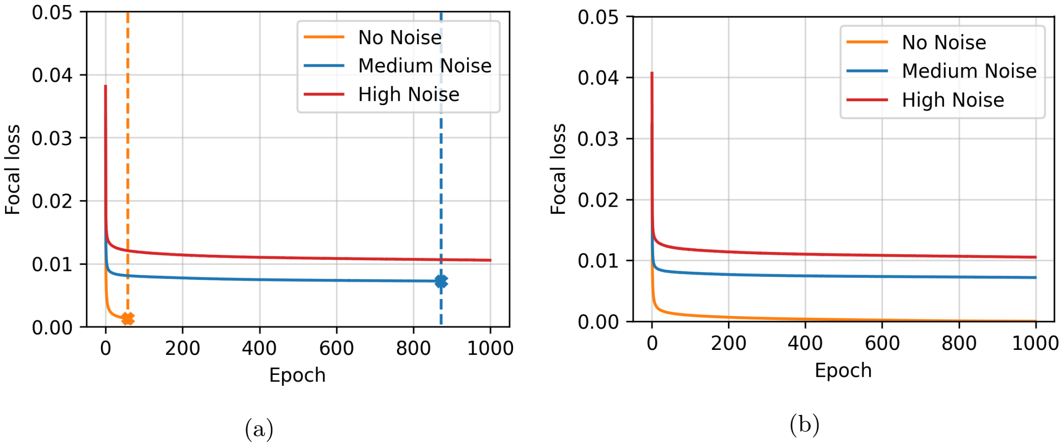


Figure 5: Results after training the 2D U-net with the Focal loss using a γ and α of 0.5 for 1000 epochs on the MNIST dataset with the original Focal loss (a), and the stabilized Focal loss (b). The model was trained without noise, and with "Medium" and "High" noise levels. A cross indicates the epoch at which a "NaN" was encountered during training.

input data delayed this point, and the instability could not be detected when enough noise was added. The segmentation results are consistent with the results from the binary classification task. Again, when using the modified version of the Focal loss, no more instabilities were reported, and all 3 models were trained to completion. Note that the losses have different asymptotes, which originate from the way that the noise was added and the segmentation masks were generated. The introduction of noise prior to generating the masks may have compromised the clarity of the boundary between the foreground and background. As a result, it becomes difficult to accurately determine this boundary after noise is introduced, leading to an increase in loss and therefore a shift in the assymptote.

4 Discussion and Conclusion

This paper addresses a hitherto unreported instability of the Focal loss when a γ value between 0 and 1 is used. We showed that this instability is not only mathematically derivable but can also be demonstrated using two simple experiments. Due to the singularity that arises in the derivative of the Focal loss when using these γ values, training deep learning models like a basic CNN classifier or 2D U-net can lead to unstable behavior. Our experiments suggest that datasets with a severe class imbalance are prone to presenting such instabilities and that the complexity of the task influences the speed at which the instability presents itself. A likely explanation is that models will overfit quickly when trained on easy tasks. This consequently results in confident predictions, causing the model output to reach the true value of the class, which results in a singularity in the derivative of the Focal loss, causing instability. The more difficult the task at hand, the more epochs the model will take to reach a state where the predictions are confident enough to trigger the instability. This is also shown when adding noise to the input data, highlighting that increasing the difficulty of that task requires more epochs to reach this critical instability point. With the presented experiments, we highlight that the instability is not necessarily an issue in all scenarios, but that it can arise under certain conditions.

To resolve this instability, we propse a modification of the original Focal loss by adding a smoothing constant to the term that downscales the cross-entropy loss. This ensures that the singularity in the Focal loss derivative is eliminated, which stabilizes model training. Where unstable γ values triggered the instability when using the original Focal loss in our experiments, the modified version completed all epochs for each experiment.

In this paper, we have provided analytical, numerical, and experimental proof of the existence of this Focal loss instability. Our simple experiments highlight that under certain conditions, the instability can be induced. We therefore recommend refraining from using γ values that fall between 0 and 1 when using the original Focal loss. If by design, the only possible values for γ fall between 0 and 1, as is the case for the Unified Focal Loss (Yeung et al., 2022), we recommend using the modified version of the Focal loss that we propose in this paper to stabilize model training. The authors who presented the Unified Focal Loss did not report any instabilities even though their loss makes use of these unstable γ values. Their published code shows clipping of the model outputs, which would prevent their model from reaching model outputs that would trigger instability. However, no explanations were provided for the clipping operation. Additionally, their paper focuses on complex segmentation tasks, which we suggest are less prone to instability. It could be possible that when their method is applied to more simplistic segmentation tasks, the instability can still occur.

In summary, we highlighted an unaddressed instability of the Focal loss and proposed a modification to prevent this instability from occurring. Our experiments showed that after modifying the Focal loss, the instabilities were effectively removed. We therefore recommend either refraining from using the unstable γ values when using the Focal loss or adopting our modification to prevent these instabilities from occurring.

References

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Mandana Hamidi and Ali Borji. Invariance analysis of modified c2 features: case study—handwritten digit recognition. *Machine Vision and Applications*, 21:969–979, 2010.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pp. 565–571. Ieee, 2016.

Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip Torr, and Puneet Dokania. Calibrating deep neural networks using focal loss. *Advances in neural information processing systems*, 33: 15288–15299, 2020.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241. Springer, 2015.

Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pp. 240–248. Springer, 2017.

Juan Terven, Diana M Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A Chavez-Urbiola, and Julio A Romero-Gonzalez. Loss functions and metrics in deep learning. *arXiv preprint arXiv:2307.02694*, 2023.

Michael Yeung, Evis Sala, Carola-Bibiane Schönlieb, and Leonardo Rundo. Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation. *Computerized Medical Imaging and Graphics*, 95:102026, 2022.

A Appendix

A.1 Derivation Focal Loss Derivative

This Appendix contains the derivations of the Focal loss derivative. We provide separate derivations for the foreground and background classes.

A.1.1 Complete Focal loss

$$\mathcal{L}_{\mathrm{F}} = -\underbrace{\alpha_t y\,(1-p)^{\gamma}\log(p)}_{\mathcal{L}_{fg}} - \underbrace{(1-\alpha_t)(1-y)\,p^{\gamma}\log(1-p)}_{\mathcal{L}_{bg}} \tag{18}$$

A.1.2 Foreground Derivative

$$\begin{aligned}\frac{d\mathcal{L}_{fg}}{dp}|_{y=1} &= -\alpha_t\left(\frac{d(1-p)^{\gamma}}{dp}\log(p) + \frac{d\log(p)}{dp}(1-p)^{\gamma}\right)\\ &= -\alpha_t\left(-\gamma(1-p)^{\gamma-1}\log(p) + \frac{(1-p)^{\gamma}}{p}\right)\\ &= \alpha_t\left(\gamma(1-p)^{\gamma-1}\log(p) - \frac{(1-p)^{\gamma}}{p}\right)\end{aligned} \tag{19}$$

A.2 Background Derivative

$$\begin{aligned}\frac{d\mathcal{L}_{bg}}{dp}|_{y=0} &= -(1-\alpha_t)\left(\frac{dp^{\gamma}}{dp}\log(1-p) + \frac{d\log(1-p)}{dp}p^{\gamma}\right)\\ &= -(1-\alpha_t)\left(\gamma p^{\gamma-1}\log(1-p) - \frac{p^{\gamma}}{1-p}\right)\end{aligned} \tag{20}$$

A.3 Modified Focal Loss

We provide code for the modified version of the original Focal loss Lin et al. (2017). Modifications to the original code are indicated by the "#Modification" comment.

```python
import torch
import torch.nn.functional as F

from torchvision.utils import _log_api_usage_once   #Modification


def sigmoid_focal_loss_modified(
    inputs: torch.Tensor,
    targets: torch.Tensor,
    alpha: float = 0.25,
    gamma: float = 2,
    reduction: str = "none",
    epsilon=1e-3 #Modification


) -> torch.Tensor:
    """
    Modified version of the Focal Loss. The epsilon scalar that is
    added to the output stabilizes the model training. Whenever
    epsilon is set to 0, it simplifies to the original Focal loss.

    Args:
        inputs (Tensor): A float tensor of arbitrary shape.
                The predictions for each example.
        targets (Tensor): A float tensor with the same shape as inputs.
                Stores the binary classification label for each element
                in inputs (0 for the negative class and
                1 for the positive class).
        alpha (float): Weighting factor in range (0,1) to balance
                positive vs negative examples or -1 for
                ignore. Default: ``0.25``.
        gamma (float): Exponent of the modulating factor (1 - p_t) to
                balance easy vs hard examples. Default: ``2``.
        epsilon(float): Smoothing constant preventing the
                instabilities when gamma values between 0 and 1
                are used. Default: ``1e-3``
        reduction (string): ``'none'`` | ``'mean'`` | ``'sum'``
                ``'none'``: No reduction will be applied to the output.
                ``'mean'``: The output will be averaged.
                ``'sum'``: The output will be summed.
                Default: ``'none'``.
    Returns:
        Loss tensor with the reduction option applied.
    """
    #Modification of the Original implementation from
    https://github.com/facebookresearch/fvcore/blob/master/fvcore/nn/focal_loss.py

    if not torch.jit.is_scripting() and not torch.jit.is_tracing():
        _log_api_usage_once(sigmoid_focal_loss_modified)  #Modification
    p = torch.sigmoid(inputs)
    ce_loss = F.binary_cross_entropy_with_logits(inputs, targets,
                reduction="none")
    p_t = (p) * targets + (1 - p) * (1 - targets)
    loss = ce_loss * ((1 - p_t+epsilon) ** gamma) #Modification

    if alpha >= 0:
        alpha_t = alpha * targets + (1 - alpha) * (1 - targets)
        loss = alpha_t * loss

    # Check reduction option and return loss accordingly
    if reduction == "none":
        pass
    elif reduction == "mean":
        loss = loss.mean()
    elif reduction == "sum":
        loss = loss.sum()
    else:
        raise ValueError(
            f"Invalid Value for arg 'reduction': '{reduction} \n
            Supported reduction modes: 'none', 'mean', 'sum'"
        )
    return loss

```

A.4 CNN

A summary of the CNN for binary classification is shown 2. The code snippet displaying the implementation of this model is provided. The CNN was modified from a CNN in a Pytorch Tutorial https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.

Table 2: Summary of the binary classification CNN

Layer (type:depth-idx)	Output Shape	Param #
CNN	[64, 1]	–
Conv2d: 1-1	[64, 6, 24, 24]	156
MaxPool2d: 1-2	[64, 6, 12, 12]	–
Conv2d: 1-3	[64, 16, 8, 8]	2,416
MaxPool2d: 1-4	[64, 16, 4, 4]	–
Linear: 1-5	[64, 120]	30,840
Linear: 1-6	[64, 84]	10,164
Linear: 1-7	[64, 1]	85

```python
import torch.nn as nn
import torch.nn.functional as F
import torch
from torchinfo import summary

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(256, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 1)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)

        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

A.5 CNN Experiment Code

This Appendix contains the code that was used to execute the experiments. In this code, the package *Simple_CNN* is the CNN shown in A.4, and the *Revised_Focal_loss* is the modified Focal loss function in Appendix A.3.

```python
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as tvt
from Simple_CNN import Net
import torch.optim as optim
import numpy as np
import pandas as pd
from tqdm import tqdm
import os
import random
import matplotlib.pyplot as plt
from Revised_Focal_loss import sigmoid_focal_loss_modified

mnist_trainset=datasets.MNIST(root='',
                train=True,download=False,
                transform=tvt.ToTensor()) #Add root where the MNIST dataset is stored


device = torch.device("cuda:0")
epsilon=1e-3
batch_size=64
epochs=100
alpha=0.5
lr=1e-3
Add_noise=True
gamma_values=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,0,1,2,3,4,5]
THs=[0,1,2,3,4,5,6,7,8]

columns = ['Gamma', 'TH']+[str(i) for i in range(0,epochs)]

noise_amplitudes=[0,0.5,0.75]

Loss_functions=['Adapted','Original']

Sigmoid=torch.nn.Sigmoid()

for Lf in Loss_functions:

    Path_to_experiments='' #insert path

    if os.path.isdir(Path_to_experiments)==False:
        os.makedirs(Path_to_experiments)

    for Na in noise_amplitudes:

        result_folder=os.path.join(Path_to_experiments,"Noise_level_%s"%Na)

        if os.path.isdir(result_folder)==False:

            All_training_losses=np.ones((len(gamma_values)*len(THs),epochs+2))*-1
            All_training_acc=np.ones((len(gamma_values)*len(THs),epochs+2))*-1

            df_epoch = pd.DataFrame (All_training_losses)
            df_ac = pd.DataFrame (All_training_acc)

            os.mkdir(result_folder)
            noise_amplitude=Na
            random.seed(123)

            Exp_nr=0
            for GAMMA in range(len(gamma_values)):
                for TH in range(len(THs)):

                    All_training_losses[Exp_nr,0]=gamma_values[GAMMA]
                    All_training_losses[Exp_nr,1]=THs[TH]
                    All_training_acc[Exp_nr,0]=gamma_values[GAMMA]
                    All_training_acc[Exp_nr,1]=THs[TH]

                    trainloader = torch.utils.data.DataLoader(mnist_trainset,
                                                             batch_size=batch_size,
                                                             shuffle=True,
                                                            num_workers=2)
                    dataiter = iter(trainloader)
                    images, labels = next(dataiter)

                    net=Net().to(device)
                    optimizer = optim.SGD(net.parameters(), lr=lr, momentum=0.9)
                    Train_loss=np.ones((1,epochs))*-1

                    for epoch in tqdm(range(epochs)):
                        running_loss = 0.0
                        train_acc=0
                        for i, data in enumerate(trainloader, 0):
                            # get the inputs; data is a list of [inputs, labels]
                            inputs, labels = data
                            for ii in range(len(labels)):
                                if labels[ii]>THs[TH]:
                                    labels[ii]=torch.tensor(0.,dtype=torch.float64)
                                    if Add_noise:
                                        noise=np.random.rand(28, 28)*noise_amplitude
                                        inputs[ii]=inputs[ii]+noise
                                        inputs[ii]=np.clip(inputs[ii],0,1)
                                else:
                                    labels[ii]=torch.tensor(1.)
                                    if Add_noise:
                                        noise=np.random.rand(28, 28)*noise_amplitude
                                        inputs[ii]=inputs[ii]+noise
                                        inputs[ii]=np.clip(inputs[ii],0,1)

                            optimizer.zero_grad()

                            inputs=inputs.to(device)

                            outputs = net(inputs)
                            labels=torch.unsqueeze(labels,1).float()
                            labels=labels.to(device)

                            if Lf=="Original":
                                loss = torchvision.ops.sigmoid_focal_loss(outputs, labels,
                                            alpha=alpha,gamma=gamma_values[GAMMA],
                                            reduction = 'mean')
                            else:
                                loss = sigmoid_focal_loss_modified(outputs, labels,
                                            alpha=alpha,gamma=gamma_values[GAMMA],
                                            reduction = 'mean',epsilon_scalar=epsilon)

                            loss.backward()
                            optimizer.step()

                            running_loss += loss.item()

                            outputs=Sigmoid(outputs)
                            outputs=torch.round(outputs)

                            train_acc += torch.sum(outputs == labels).item()

                        Epoch_accuracy=train_acc/len(mnist_trainset)

                        Epoch_loss= running_loss/len(trainloader)

                        if torch.isnan(loss)==True:
                            print('Terminated due to NaN at epoch %s'%epoch)
                            df_epoch.iloc[Exp_nr,epoch+2]='inf'
                            df_ac.iloc[Exp_nr,epoch+2]='inf'

                            break
                        else:
                            df_epoch.iloc[Exp_nr,epoch+2]=Epoch_loss
                            df_ac.iloc[Exp_nr,epoch+2]=Epoch_accuracy


                    print('Finished Training')

                    df_epoch.iloc[Exp_nr,:]=df_epoch.iloc[Exp_nr,:].replace(-1,'inf')
                    df_ac.iloc[Exp_nr,:]=df_ac.iloc[Exp_nr,:].replace(-1,'inf')

                    df_epoch.columns=columns
                    df_ac.columns=columns

                    filepath_epochs = os.path.join(result_folder,
                                    'Experiment_Results_no_noise_new_loss_epoch.xlsx')
                    filepath_accuracy = os.path.join(result_folder,
                                    'Experiment_Results_no_noise_new_loss_acc.xlsx')

                    df_epoch.to_excel(filepath_epochs, index=False)
                    df_ac.to_excel(filepath_accuracy, index=False)

                    Exp_nr+=1
```

A.6 Unet

The U-net used for this paper was obtained from `https://github.com/clemkoa/u-net/blob/master/unet/unet.py` with some minor modification.

Table 3: Summary of the 2D U-Net architecture

Layer (type:depth-idx)	**Output Shape**	**Param #**
UNet	[64, 1, 28, 28]	–
Conv2d: 3-1	[64, 64, 28, 28]	640
BatchNorm2d: 3-2	[64, 64, 28, 28]	128
Conv2d: 3-4	[64, 64, 28, 28]	36,928
BatchNorm2d: 3-5	[64, 64, 28, 28]	128
Sequential: 3-7	[64, 128, 14, 14]	221,952
Sequential: 3-8	[64, 256, 7, 7]	886,272
Sequential: 3-9	[64, 512, 3, 3]	3,542,016
Sequential: 3-10	[64, 1024, 1, 1]	14,161,920
ConvTranspose2d: 3-11	[64, 512, 2, 2]	2,097,664
Sequential: 3-12	[64, 512, 3, 3]	7,080,960
ConvTranspose2d: 3-13	[64, 256, 6, 6]	524,544
Sequential: 3-14	[64, 256, 7, 7]	1,771,008
ConvTranspose2d: 3-15	[64, 128, 14, 14]	131,200
Sequential: 3-16	[64, 128, 14, 14]	443,136
ConvTranspose2d: 3-17	[64, 64, 28, 28]	32,832
Sequential: 3-18	[64, 64, 28, 28]	110,976
Conv2d: 1-10	[64, 1, 28, 28]	65
Total params		31,042,369
Trainable params		31,042,369
Non-trainable params		0
Total mult-adds (G)		36.16
Input size (MB)		0.20
Forward/backward size (MB)		425.34
Params size (MB)		124.17
Estimated Total Size (MB)		549.71

```python
#Downloaded and modified from:
#https://github.com/clemkoa/u-net/blob/master/unet/unet.py

import torch
from torch import nn
import torch.nn.functional as F

class DoubleConv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_ch, out_ch, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        x = self.conv(x)
        return x

class Up(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(Up, self).__init__()
        self.up_scale = nn.ConvTranspose2d(in_ch, out_ch,
        kernel_size=2, stride=2)

    def forward(self, x1, x2):
        x2 = self.up_scale(x2)

        diffY = x1.size()[2] - x2.size()[2]
        diffX = x1.size()[3] - x2.size()[3]

        x2 = F.pad(x2, [diffX // 2, diffX - diffX // 2,
        diffY // 2, diffY - diffY // 2])
        x = torch.cat([x2, x1], dim=1)
        return x


class DownLayer(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(DownLayer, self).__init__()
        self.pool = nn.MaxPool2d(2, stride=2, padding=0)
        self.conv = DoubleConv(in_ch, out_ch)

    def forward(self, x):
        x = self.conv(self.pool(x))
        return x


class UpLayer(nn.Module):
    def __init__(self, in_ch, out_ch):
        super(UpLayer, self).__init__()
        self.up = Up(in_ch, out_ch)
        self.conv = DoubleConv(in_ch, out_ch)

    def forward(self, x1, x2):
        a = self.up(x1, x2)
        x = self.conv(a)
        return x


class UNet(nn.Module):
    def __init__(self,channels=1, dimensions=1):
        super(UNet, self).__init__()
        self.conv1 = DoubleConv(channels, 64)
        self.down1 = DownLayer(64, 128)
        self.down2 = DownLayer(128, 256)
        self.down3 = DownLayer(256, 512)
        self.down4 = DownLayer(512, 1024)
        self.up1 = UpLayer(1024, 512)
        self.up2 = UpLayer(512, 256)
        self.up3 = UpLayer(256, 128)
        self.up4 = UpLayer(128, 64)
        self.last_conv = nn.Conv2d(64, dimensions, 1)

    def forward(self, x):
        x1 = self.conv1(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x1_up = self.up1(x4, x5)
        x2_up = self.up2(x3, x1_up)
        x3_up = self.up3(x2, x2_up)
        x4_up = self.up4(x1, x3_up)
        output = self.last_conv(x4_up)
        return output

```

A.7 Unet Experiment Code

Code to run the segmentation experiments.

```python

from Revised_Focal_loss import sigmoid_focal_loss_revised
from unet import UNet
import torch
import torch.optim as optim
import torchvision
import torchvision.datasets as datasets
import copy
import os
from torch.utils.data import DataLoader
import numpy as np
import pandas as pd

device = torch.device("cuda:1")

path_to_results="" # insert path to results

if os.path.isdir(path_to_results)==False:
    os.makedirs(path_to_results)

transforms = torchvision.transforms.Compose([
torchvision.transforms.ToTensor(),

mnist_trainset=datasets.MNIST(root='',
            train=True,download=False,
            transform=transforms) #Add root where the MNIST dataset is stored

batch_size=64
epochs=1000
gamma=0.5
alpha=0.5
epsilon=1e-3 # 0 for original Focal loss
train_loader = torch.utils.data.DataLoader(mnist_trainset,
                        batch_size=batch_size,
                        shuffle=True,
                        num_workers=2)

Noise_levels=[0,0.5,0.75]
running_loss=0.0

columns = ['Noise']+[str(i) for i in range(0,epochs)]
All_training_losses=np.ones((len(Noise_levels),epochs+1))*-1
df_epoch = pd.DataFrame (All_training_losses)

Epochs_losses=[]
Experiment_nr=0

for nl in range(len(Noise_levels)):

    Unet=UNet(channels=1).to(device)
    optimizer = optim.SGD(Unet.parameters(), lr=0.001, momentum=0.9)

    NA=Noise_levels[nl]
    All_training_losses[Experiment_nr,0]=NA

    for epoch in range(epochs):

        running_loss=0.0
        batch=0

        for inputs, labels in train_loader:

                optimizer.zero_grad()

                noise=torch.tensor(np.random.rand(len(inputs),1,28, 28)*NA)

                labels=copy.copy(inputs)>0.5
                labels=labels.type(torch.float)
                labels=labels.to(device)

                inputs=inputs+noise
                inputs=np.clip(inputs,0.0,1.0).type(torch.float)
                inputs=inputs.to(device)
                outputs = Unet(inputs)

                loss= sigmoid_focal_loss_revised(outputs,
                    labels,alpha=alpha,
                    gamma=gamma,
                    reduction = 'mean',
                    epsilon_scalar=epsilon)

                loss.backward()
                optimizer.step()
                running_loss += loss.item()

        running_loss=running_loss/len(train_loader)
        Epochs_losses.append(running_loss)

        if torch.isnan(loss)==True:
            print('Terminated due to NaN at epoch %s'%epoch)
            df_epoch.iloc[Experiment_nr,epoch+1]='inf'

            break
        else:
            df_epoch.iloc[Experiment_nr,epoch+1]=running_loss

    Experiment_nr+=1
    print('Finished Training')

df_epoch.iloc[:,:]=df_epoch.iloc[:,:].replace(-1,'inf')
df_epoch.columns=columns

filepath_epochs = os.path.join(path_to_results,
                'Segmentation_results.xlsx')

df_epoch.to_excel(filepath_epochs, index=False)

```