This is an Article: It has a long subtitle containing important SEO keywords

Min-Seo Kim

{kms39273@catholic.ac.kr}

# ABSTRACT

Abstract— This document is a template for Microsoft Word for the submission of a multi-page paper to the journal. sheet, as illustrated by the portions given in this document.

Keywords: write 3-5 descriptive keywords here

목차

# EDA

## Understanding the problem and the dataset thoroughly before conducting deep learning training is crucial for verifying data quality, understanding data distribution, selecting an appropriate model, and devising preprocessing methods.

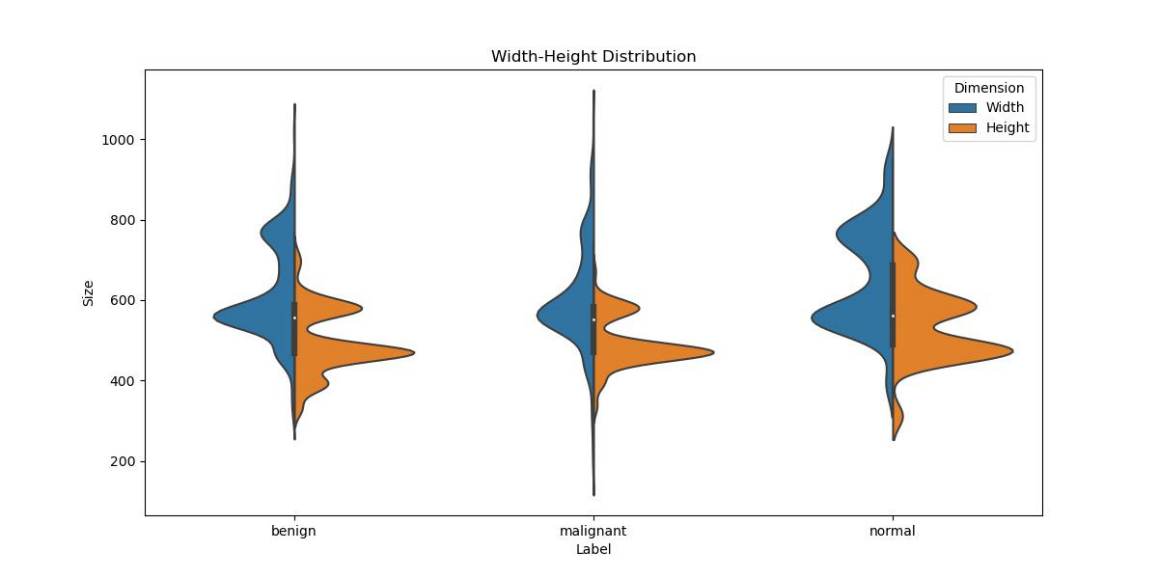
## A. Problem Definition

Breast cancer is one of the leading causes of death among women, making early detection crucial. This project aims to develop a model that classifies breast cancer using ultrasound images.

## B. About the Dataset

This dataset includes breast ultrasound images collected from 600 women aged between 25 and 75 in 2018, comprising a total of 780 images in PNG format. The images have an average size of 500x500 pixels and are categorized into three classes: normal, benign, and malignant.

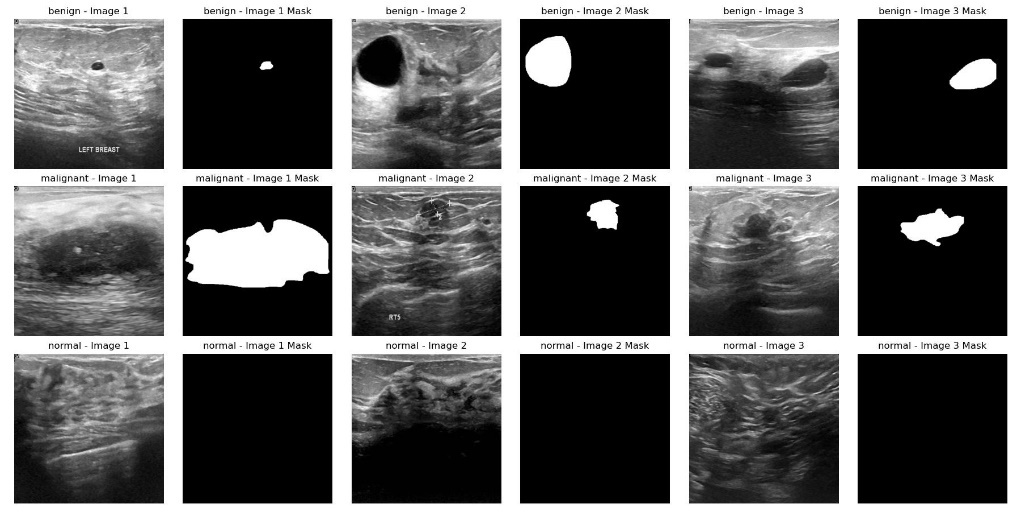
## C. Checking Image Size Distribution

Firstly, from the dataset description, we can infer that the sizes of the images vary. Therefore, we checked the dimensions of the images to understand their size distribution.

After calculating the mean and quartiles of the width and height, it was found that approximately 50% of the data falls within the range of 500-700 pixels for width and 450-600 pixels for height.

The dataset contains 891, 421, and 266 files for each of the three classes, respectively. Due to the variation in the number of samples per class, methodologies such as stratified sampling are required during training.

Additionally, the overall number of samples per class is relatively small, which may necessitate the use of data augmentation techniques. Furthermore, employing k-fold cross-validation could be beneficial to improve model reliability.

 Visually, it is observed that no tumors are present in the normal class images. In contrast, tumors can be seen in both the malignant and benign class images. Additionally, there appears to be a difference in the degree of clarity between the malignant and benign classes, with malignant images generally exhibiting a higher level of clarity.

## D. Checking Image Size Distribution

To facilitate future normalization, we also calculated the mean and standard deviation of pixel values for each channel.

Class: Benign

Files found: 437

Mean of pixel values: 86.28

Standard deviation of pixel values: 8.82

Class: Malignant

Files found: 210

Mean of pixel values: 79.77

Standard deviation of pixel values: 9.29

Class: Normal

Files found: 133

Mean of pixel values: 81.01

Standard deviation of pixel values: 9.45

Overall Statistics

Mean of pixel values (All classes): 83.63

Standard deviation of pixel values (All classes): 9.16

## E. Summary

The main characteristics of the dataset observed are as follows:

* The number of data points per class is small and imbalanced.
* There is a variance in the size of the images.

Considering these characteristics, a baseline model was developed.

# II. Baseline Model Implementation

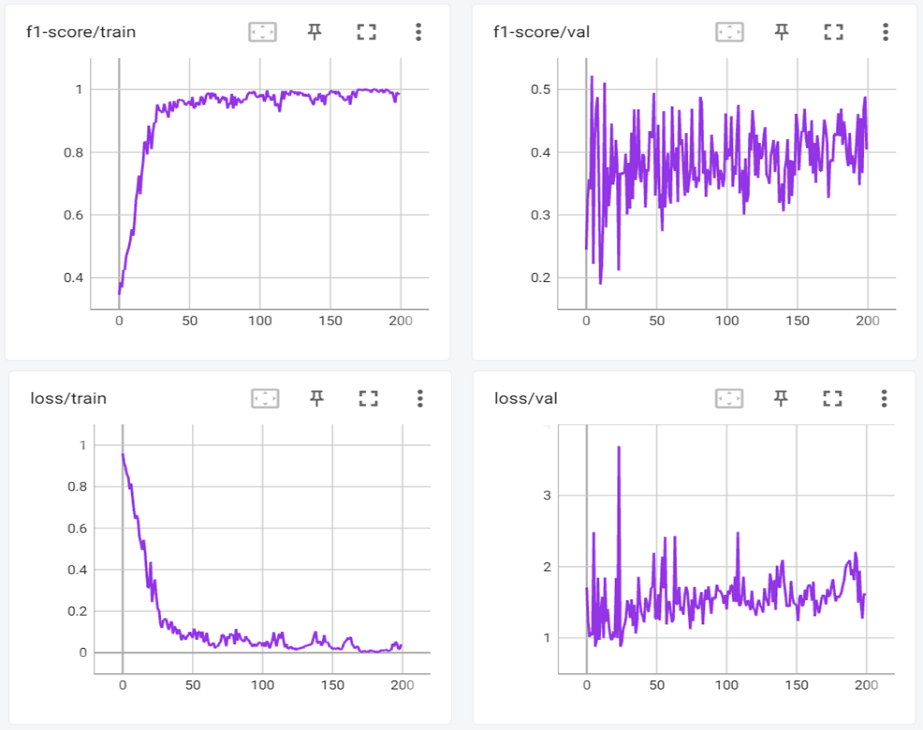
## A. Experimental Setup

Due to the limited number of data points per class, performance evaluation will be conducted using k-fold cross-validation instead of a simple train-validation-test split. Given the class imbalance, we employed stratified k-fold cross-validation to ensure the class distribution is maintained in both the training and validation sets.

To optimize training speed, the images were converted to grayscale and rescaled to 300x300 pixels.

## B. Initial Model Implementation

For the initial model implementation, ResNet-101 was used as the baseline model. The learning rate was set to 0.00005, and a batch size of 16 was chosen. The Adam optimizer was employed for training the model, with the ReLU function as the activation function. The weights were initialized using the Kaiming initialization method.

After conducting the experiment, the F-1 score and loss were used as performance metrics. The observation of the validation loss diverging indicated that the model was likely overfitting.

# III. Addressing Overfitting

## A. data augmentaion

Data augmentation involves creating new training samples by applying various transformations to the existing data. This helps which can improve the model's ability to generalize and prevent overfitting. By exposing the model to a wider variety of data, augmentation helps it learn more robust features that are less sensitive to variations in the input.

Data augmentation is particularly useful when dealing with limited datasets, as it provides a way to generate more training examples without actually collecting more data. This technique helps improve the model's robustness and generalization capability, reduces overfitting, and can lead to better performance on unseen data.

Random Horizontal Flip

Randomly flips the image horizontally with a specified probability (e.g., 50%). This helps the model learn invariant features to horizontal flips.

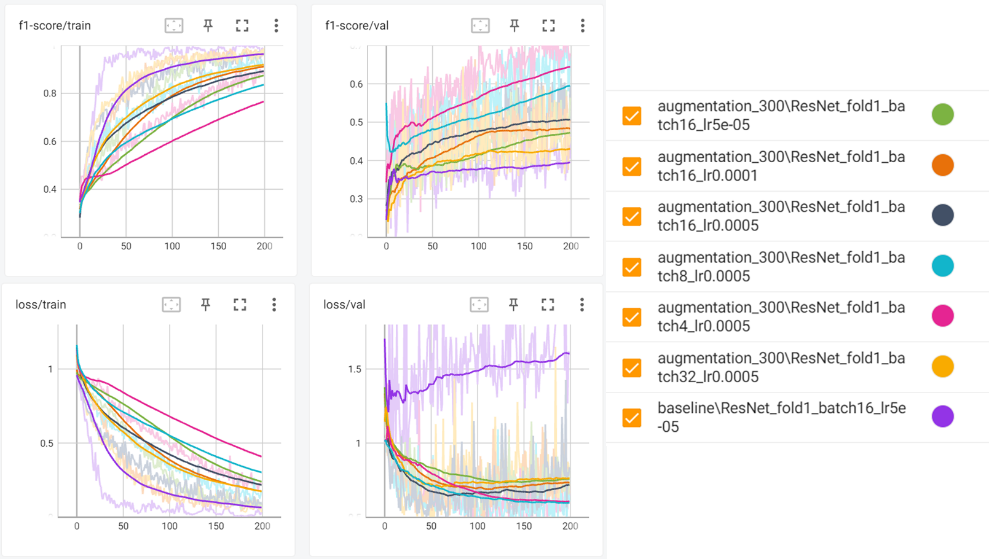
Random Rotation

Randomly rotates the image by a certain angle (e.g., up to 30 degrees). This helps the model become invariant to rotational variations in the input data.

Normalization

Adjusts the pixel values of the image to have a mean of 0 and a standard deviation of 1. This standardization helps improve the convergence speed of the model and ensures that the data is on a similar scale.

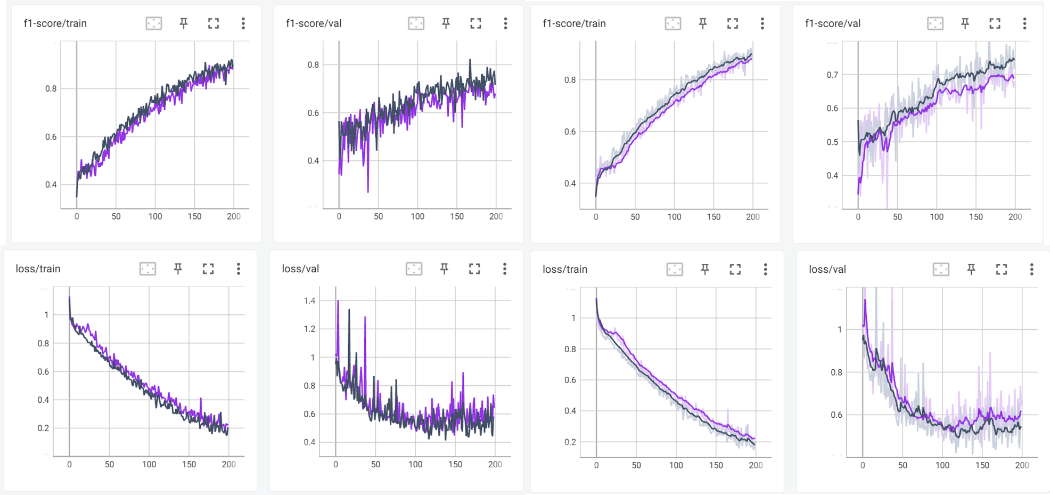
RandomHorizontalFlip 50%, RandomRotation(30), and data normalization were conducted. Subsequently, learning rates from 1e-4 to 5e-5 and batch sizes between 2 and 32 were tested.

 Through the application of data augmentation, we observed a noticeable improvement in mitigating overfitting. Specifically, as the batch size decreased, the model's generalization performance improved. Furthermore, we determined that a learning rate of 5e-4 was the sweet spot for our experiments, providing an optimal balance between convergence speed and performance.

The subsequent experiments were all conducted with a batch size of 4, and a default learning rate of 5e-4.

# IV. Performance Improvement

## A. Weight Initialization(Xavier vs Kaiming)

To understand the impact of weight initialization methods on model performance, an experiment was conducted using the same learning rate of 5e-4 and a batch size of 4. The two initialization methods compared were Xavier and Kaiming.

The results indicated that **Xavier initialization (shown in dark blue)** demonstrated a slight performance advantage over **Kaiming initialization(shown in purple)** . This suggests that Xavier initialization might provide a better starting point for training the model, potentially leading to faster convergence and improved performance. The superior performance with Xavier initialization could be due to its ability to maintain the variance of activations through layers, which is crucial for training deep networks effectively.

All subsequent experiments used the Xavier initialization method.

## B. Compare activation function

Activation functions introduce non-linearity into the neural network, enabling it to learn complex patterns and representations. Without activation functions, the network would only be able to learn linear mappings, severely limiting its capacity to model real-world data.

Different activation functions have different properties and can affect the performance and convergence of the neural network. Some activation functions might work better for certain types of data or architectures, leading to faster training and improved accuracy. Experimenting with various activation functions helps in identifying the most suitable one for the given task, thereby enhancing model performance.

**Sigmoid**

The Sigmoid activation function maps any input value to a value between 0 and 1. It is commonly used in binary classification problems. However, it can suffer from the vanishing gradient problem, making it less effective for deep networks.

**ReLU (Rectified Linear Unit)**

ReLU activation function outputs the input directly if it is positive; otherwise, it outputs zero. It is widely used due to its simplicity and effectiveness in avoiding the vanishing gradient problem. However, it can suffer from the "dying ReLU" problem where neurons can become inactive.

**Leaky ReLU**

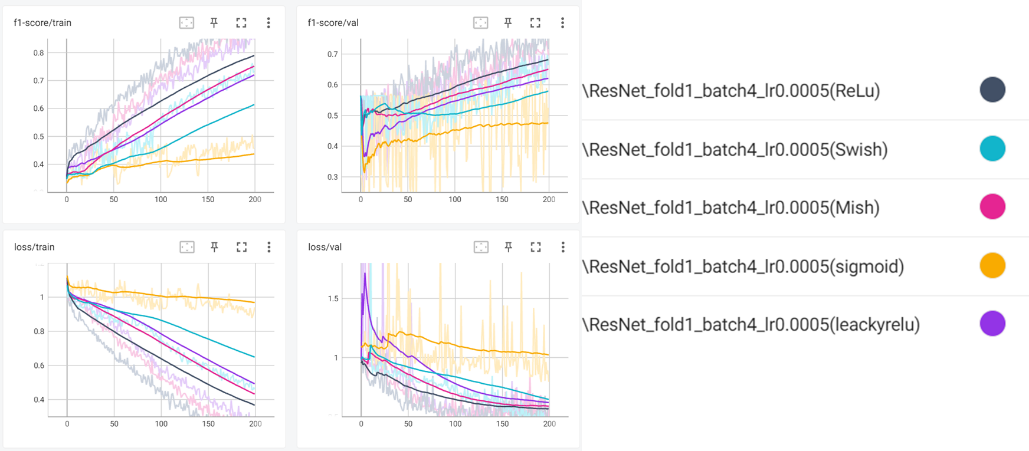
Leaky ReLU is a variant of ReLU that allows a small, non-zero gradient when the input is negative. This helps to prevent the dying ReLU problem by ensuring that neurons can recover from inactivity.

**Swish**

Swish is a newer activation function defined as ． It has been found to perform better than ReLU on deeper networks and has the advantage of being smooth and non-monotonic.

**Mish**

Mish is another newer activation function defined as ． It is smooth and non-monotonic, like Swish, and has been shown to improve performance on a variety of deep learning tasks by providing better gradients during training.

**Experimental Results**

The superior performance of ReLU can be attributed to its simplicity and efficiency in deep networks. ReLU effectively avoids the vanishing gradient problem, leading to faster convergence and better overall performance.

Mish and Leaky ReLU performed well because they offer smoother gradients and the ability to recover from neuron inactivity, respectively. Mish, being a newer activation function, combines the benefits of non-linearity and smoothness, which can lead to better gradient flow and performance in deep networks.

Swish, while also a smooth and effective activation function, performed slightly worse than Mish and Leaky ReLU, possibly due to differences in how it scales and impacts gradient flow.

Sigmoid performed the worst due to its tendency to suffer from the vanishing gradient problem, which can slow down learning and affect the performance of deeper networks.

The subsequent experiments were all conducted using the ReLU activation function.

## C. Compare optimizers

Optimizers play a critical role in training neural networks. They are algorithms or methods used to adjust the weights and biases of the network in order to minimize the loss function. The loss function measures how well the model's predictions match the actual data, and the optimizer helps in finding the set of parameters (weights and biases) that result in the lowest possible loss.

Different optimizers have different strategies for updating the weights and biases. These strategies can significantly impact the speed of convergence, the stability of training, and the final performance of the model. Therefore, experimenting with different optimizers can help identify the one that best suits the specific characteristics of the dataset and the model architecture.

**SGD (Stochastic Gradient Descent)**

SGD updates the weights based on the gradient of the loss function with respect to the parameters for a single training sample at a time. While simple and effective, it can be slow and may get stuck in local minima.

**Momentum**

Momentum builds on SGD by adding a fraction of the previous weight update to the current update. This helps accelerate gradients vectors in the right directions, leading to faster converging and reduced oscillation.

**RMSprop (Root Mean Square Propagation)**

RMSprop modifies the update rule of SGD by dividing the learning rate by an exponentially decaying average of squared gradients. This helps to normalize the updates and can lead to faster convergence and better performance on non-stationary problems.

**Adam (Adaptive Moment Estimation)**

Adam combines the benefits of both Momentum and RMSprop. It calculates adaptive learning rates for each parameter and includes momentum, which helps in achieving faster convergence and better performance in practice.

**Nadam (Nesterov-accelerated Adaptive Moment Estimation)**

텍스트, 도표, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명Nadam is an extension of Adam that incorporates Nesterov momentum. This can lead to even faster convergence and potentially better performance compared to Adam.

To compare the performance of different optimizers, experiments were conducted using SGD, Momentum, RMSprop, Adam, and Nadam. Among these, Adam converged with comparatively lower loss values. Interestingly, the Momentum optimizer also demonstrated strong performance, despite showing some fluctuations in the validation loss.

Based on these observations, the subsequent experiments will be conducted using both the Momentum and Adam optimizers. This approach leverages the stability and performance of Adam, alongside the promising results seen with Momentum, to ensure comprehensive evaluation and robust model performance.

## D. Micro vs Macro vs Weighted

To make efficient comparisons, we explored and applied different methods, although we have been using the macro method up to now.

**Micro**

This method treats the predictions of all classes as if they belong to a single class, summing up the total TP (True Positive), FP (False Positive), and FN (False Negative) to calculate performance.

Characteristics: Aggregates the prediction results of all classes.

Advantages: Treats each sample equally, even in the presence of class imbalance.

Disadvantages: The performance of classes with a larger number of samples can dominate the overall result.

**Macro**

This method calculates the performance for each class separately and then takes the simple average.

Characteristics: Considers the performance of each class equally.

Advantages: Emphasizes the performance of each class equally.

Disadvantages: In the presence of class imbalance, the performance of classes with fewer samples can be overestimated.

**Weighted**

This method calculates the performance of each class and then computes the weighted average based on the number of samples in each class.

Characteristics: Gives more weight to the performance of classes with more samples in case of class imbalance.

Advantages: Reflects the actual performance more accurately in datasets with class imbalance.

Disadvantages: The performance of classes with fewer samples may be relatively underemphasized.

**Summary**

Micro: Calculates performance by treating all data as belonging to a single class.

Macro: Calculates the simple average of each class's performance, treating them equally.

Weighted: Computes a weighted average of each class's performance based on the number of samples.

텍스트, 도표, 그래프, 라인이(가) 표시된 사진

자동 생성된 설명The experiments were conducted in a ReLU environment with a batch size of 4 and a learning rate of 0.005.

In our case, the performance for the benign class, which has a larger number of samples, is relatively higher. Due to the class imbalance in the dataset, the Weighted method, which calculates the weighted average based on the number of samples, showed the best performance. Additionally, because the Weighted method better reflects performance in imbalanced datasets, we will continue to use the Weighted method for representing performance metrics moving forward.

## E. Detailed Comparison of Optimizers: Adam vs Momentum

Based on the previous comparison where Adam and Momentum optimizers showed the best performance, a more detailed comparison was conducted to evaluate their differences further. This experiment included three setups; Adam, Momentum with a factor of 0.9, Momentum with a factor of 0.8

텍스트, 도표, 그래프, 라인이(가) 표시된 사진

자동 생성된 설명

The results indicated that Momentum with a factor of 0.9 had a slight edge, followed by Momentum with a factor of 0.8, and then Adam. However, the performance differences were minimal.

Given the close performance results, it was decided that future experiments will take these findings into account without committing to a single optimizer. Both Adam and Momentum will be used as needed, ensuring flexibility in optimizing different aspects of the model.

## F. Learning Rate Scheduler

A learning rate scheduler dynamically adjusts the learning rate during training. This helps in finding an optimal learning rate that can lead to faster convergence and improved performance. By reducing the learning rate when the model reaches a plateau, the scheduler helps in fine-tuning the weights more effectively, avoiding overshooting the optimal point.

In my experiments, I tested two different learning rate environments with and without the application of a learning rate scheduler:

1. Batch size: 4, Learning rate: 0.005 (with and without scheduler)

2. Batch size: 4, Learning rate: 0.0005 (with and without scheduler)

This resulted in a total of four experimental setups

텍스트, 도표, 그래프, 라인이(가) 표시된 사진

자동 생성된 설명

The results showed that in the environment with a higher learning rate of 0.005, the learning rate scheduler had a positive effect. The scheduler improved performance to levels comparable to the lower learning rate environment of 0.0005 in terms of F1 score and loss.

However, in the environment with the lower learning rate of 0.0005, the learning rate scheduler did not provide any noticeable benefit. The performance remained similar to that without the scheduler.

This suggests that the learning rate scheduler is particularly useful in scenarios where the initial learning rate is relatively high. It helps in gradually reducing the learning rate, thereby allowing the model to fine-tune more effectively and avoid overshooting, leading to improved performance. On the other hand, when the learning rate is already low, the benefits of further reducing it are minimal, hence the scheduler does not show much effect.

## G. CrossEntropyLoss vs MultiMarginLoss

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명In an attempt to enhance performance using the latest techniques, I experimented with Margin Loss. Specifically, I compared CrossEntropyLoss with MultiMarginLoss.

After observing the results for approximately 70 epochs, it became clear that the performance improvement was negligible. Consequently, I decided to stop the experiment.

Despite the initial hypothesis that MultiMarginLoss would outperform CrossEntropyLoss, the null hypothesis could not be rejected. This indicates that MultiMarginLoss did not show a significant advantage over CrossEntropyLoss in this particular experiment.

Certain loss functions might interact differently with various model architectures. It's possible that the architecture of the neural network used in this experiment did not complement the MultiMarginLoss as effectively as it did with CrossEntropyLoss.

# IV. Result

텍스트, 스크린샷, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

Throughout approximately 40 search iterations, extensive experimental data was gathered and analyzed to optimize the model's performance.

텍스트, 도표, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명

As a result, an F1 score exceeding 90% was achieved. During this process, several key factors and techniques were identified as instrumental in improving the model's performance.

Initially, the experiments began with a baseline model using ResNet-101, with a batch size of 4 and a learning rate of 0.005. The Adam optimizer and ReLU activation function were employed. This setup provided a foundational understanding of the model's capabilities and areas for improvement.

A detailed comparison of different optimizers, including SGD, Momentum, RMSprop, Adam, and Nadam, was then conducted. Adam and Momentum demonstrated superior performance, prompting further experiments to focus on these two optimizers. It was found that Momentum with a factor of 0.9 had a slight performance edge, followed by Momentum with a factor of 0.8, and then Adam. However, the differences were minimal, leading to the decision to use both Adam and Momentum in future experiments for flexibility.

The impact of learning rate schedulers was also evaluated. Experiments showed that while a lower learning rate of 0.0005 did not benefit significantly from a scheduler, a higher learning rate of 0.005 saw substantial improvements when a scheduler was applied. This indicated that learning rate schedulers are particularly beneficial in scenarios with higher initial learning rates, helping to fine-tune the model more effectively.

Moreover, various activation functions were tested, including ReLU, Mish, Leaky ReLU, Swish, and Sigmoid. ReLU performed the best, followed by Mish, Leaky ReLU, Swish, and Sigmoid. ReLU's simplicity and efficiency in avoiding the vanishing gradient problem contributed to its superior performance.

Data augmentation techniques were also applied to address overfitting. Random horizontal flipping, random rotation, and data normalization were employed. These techniques improved the model's generalization performance, especially with smaller batch sizes. The experiments revealed that a learning rate of 5e-4 was optimal, striking a balance between convergence speed and performance.

A comparison between CrossEntropyLoss and MultiMarginLoss was conducted, with CrossEntropyLoss proving to be more effective. Although the hypothesis was that MultiMarginLoss would outperform CrossEntropyLoss, the experiments showed negligible performance improvements with MultiMarginLoss, leading to the conclusion that CrossEntropyLoss remained the preferred choice.

# VI. Additional Transfer Learning Experiment

In addition to the previous experiments, I conducted a transfer learning experiment using pretrained models from the torchvision library. The pretrained models were fine-tuned on the dataset.

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

The results were striking!: the performance achieved in approximately 20 epochs with transfer learning matched the performance achieved in 200 epochs without transfer learning. This demonstrates that transfer learning is highly effective in accelerating the training process and achieving comparable performance in significantly fewer epochs.

Overall, the use of transfer learning proved to be a valuable approach for rapid and efficient model training.