Long-Tailed Image Classification based on ConvNeXt-Tiny Model

RENGHE TANG* and WEIJIA LYU*, University of British Columbia, CA

In this study, we focus on the Caltech-101 dataset, which is a long-tailed (imbalanced) dataset, and use the convNeXt-tiny network for image classification. We utilize a layered sampling technique to divide the dataset, allocating 70% for training, 15% for validation, and the remaining 15% for testing purposes. This allows us to create representative validation and test sets without disrupting the original dataset distribution. To alleviate the data imbalance issue, we use weighted random sampling on the training set to ensure that each sample has an equal chance of being selected. To address overfitting, we employ online data augmentation techniques, mixup, and group L2 weight decay. We also reduce loss jitter by using exponential moving average (EMA) techniques to improve model robustness and generalization on test data. Throughout the training process, we employ a cosine annealing learning rate schedule. This involves a higher learning rate in the initial phases for quicker convergence, followed by a reduced learning rate in later stages to improve accuracy. We choose to use weighted F1 score[1], Matthew's correlation coefficient (MCC)[2], and Cohen's Kappa score[3] to evaluate model performance. These metrics weight each class, alleviating the impact of dataset imbalance on accuracy. Finally, we leverage transfer learning by initializing our model using weights pre-trained on the ImageNet-11k dataset, which allows us to explore the potential performance of ConvNeXt-Tiny model. Our objective is to utilize the ConvNeXt-Tiny model and some of the tricks proposed in the ConvNeXt[4] paper, while incorporating tricks for combating imbalanced datasets to alleviate the problem of low accuracy for small sample categories in long-tailed datasets.

1 INTRODUCTION

The classification of images plays a crucial role in the domain of computer vision, which aims to determine which specific category an image belongs to. Image classification has wide applications in real-life scenarios such as autonomous driving, medical image classification, and more [5]. As convolutional neural networks (CNNs) continue to evolve and advance [6], from AlexNet to the current ConvNeXt, the accuracy of image classification has been continuously improving. However, in the face of imbalanced (long-tailed) datasets, image classification is still not accurate enough for small sample categories [7]. In long-tailed imbalanced datasets, the number of samples in some categories is much larger than that in other categories, which may cause the model to not fully learn the features of small sample categories, leading to a decrease in performance [8].

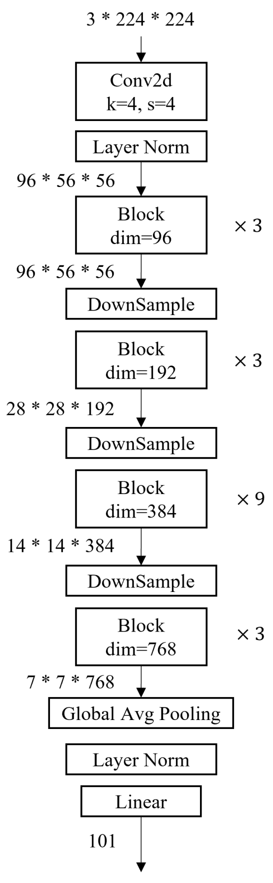
The goal of this paper is to use the convNeXt-tiny network for image classification on long-tailed imbalanced datasets and adopt corresponding strategies to address overfitting and the problems brought by long-tailed data. To enhance the model's performance, online data augmentation techniques[9] were employed. These techniques included random horizontal flipping, random vertical flipping, random rotation, random cropping and scaling, and random adjustment of brightness, contrast, saturation, and hue. At the same time, we introduce mixup techniques[10] and group L2 weight decay[11]to alleviate the overfitting phenomenon. In addition, we use exponential moving average (EMA) techniques[12] to reduce loss jitter, thereby improving the model's robustness and generalization on test data. To further optimize the training process, we adopt cosine annealing learning rate scheduling strategies[13].

2 BACKGROUND

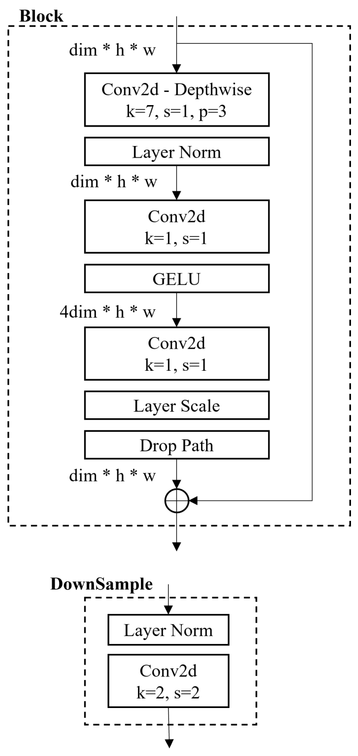
2.1 ConvNeXt-Tiny Model

The ConvNeXt model has borrowed many excellent structures from other models and used lots of training tricks. The model structure is shown in Figure 1.

*Both authors contributed equally to this research.



(a) Overall Structure



(b) Block Structure

Fig. 1. ConvNext Model Architecture[14]

As shown in Figure 1, in terms of structure, ConvNeXt incorporates grouped convolutions, with each block representing a group of convolutions. For convNeXt-tiny, the proportions of the four blocks are borrowed from Swin-tiny, with a ratio of 1:1:3:1. Additionally, it imitates Swin-Transformer to increase the number of channels in the first block to 96, and splits the downsampling layer into a layer norm and a Conv2D. It also replaces the ReLU optimizer with GeLU and reduces the use of batch normalization and activation functions.

Furthermore, many training tricks are utilized, such as using warmup and cosine annealing strategies to adjust the learning rate, and employing more sophisticated data augmentation strategies, including Mixup, CutMix, Random Erasing, RandAugment, and more.

We chose ConvNeXt-Tiny, which is a smaller version of the ConvNeXt model that is suitable for limited computing resources. Although its stage ratio is smaller compared to the base model, ConvNeXt-tiny still exhibits impressive performance on ImageNet. We believe that it will also perform well on the long-tailed dataset involved in this study.

In our implementation, we used the code from WZMIAOMIAO and Facebook AI Research [15, 16].

2.2 Caltech-101 Dataset

The dataset used in this study is Caltech-101[17], which is a long-tailed imbalanced dataset consisting of 101 categories with approximately 9000 images. The distribution of each category of images is shown in Figure 2.

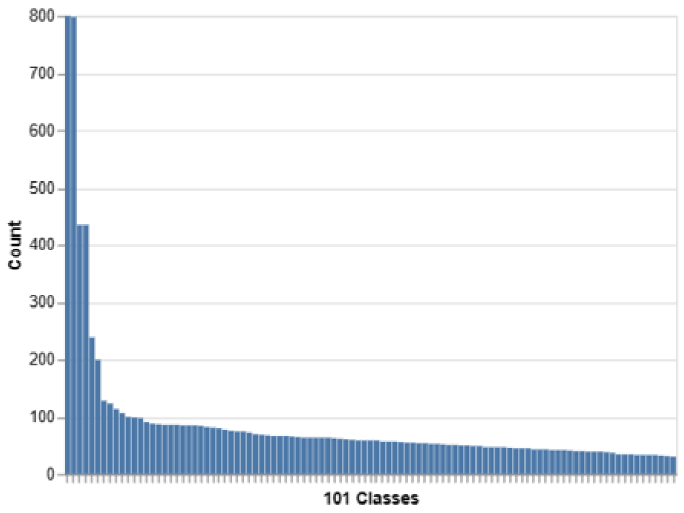


Fig. 2. Categories Distribution of Caltech-101

A long-tailed dataset means that there is a significant difference in sample size between different categories. This may cause the network couldn't learn enough from small sample categories.

2.3 Metrics

The model performance was chosen to be evaluated using Weighted F1 score[1], Matthew's correlation coefficient (MCC)[2], and Cohen's Kappa score[3]. These metrics were employed to weight each class in order to mitigate the impact of imbalanced class sample sizes on accuracy.

3 METHODOLOGY

3.1 Sampling Method

Weighted random sampling is used, which assigns different weights to each category based on the number of images (categories with more images are assigned lower weights, while categories with fewer images are assigned higher weights), such that the expected probability of sampling each category is equal to 1. This helps the model to learn small sample categories more fully and alleviates the class imbalance problem.

3.2 Data Augmentation

To improve the model's generalization ability, we employ online data augmentation techniques. Specifically, we randomly apply horizontal flipping, vertical flipping, rotation (-20° to 20°), cropping and scaling (0.8 to 1.2 times), and random adjustments of brightness, contrast, saturation, and hue to the training images, as shown in Figure 3.

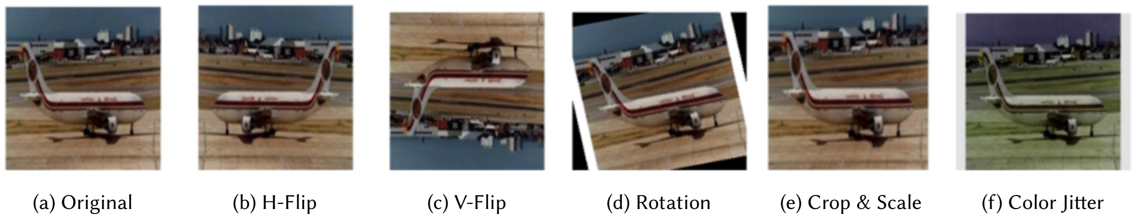


Fig. 3. Five RandAugment Techniques

By applying these operations, we generate a large number of new images, which increases the diversity of the training data and helps alleviate overfitting.

3.3 Mixup

Mixup technique[18] is a data augmentation method that mixes two images by linear interpolation. Given two image samples image1 and image2, and a random number $\lambda \in [0, 1]$ generated by beta distribution, we mix the two images using equation (1).

$$new\_img = \lambda \times img1 + (1 - \lambda) \times img2 \tag{1}$$

The loss value of the new images is calculated using equation (2).

$$new\_loss = \lambda \times loss\_img1 + (1 - \lambda) \times loss\_img2 \tag{2}$$

The generated new samples are shown in Figure 4.

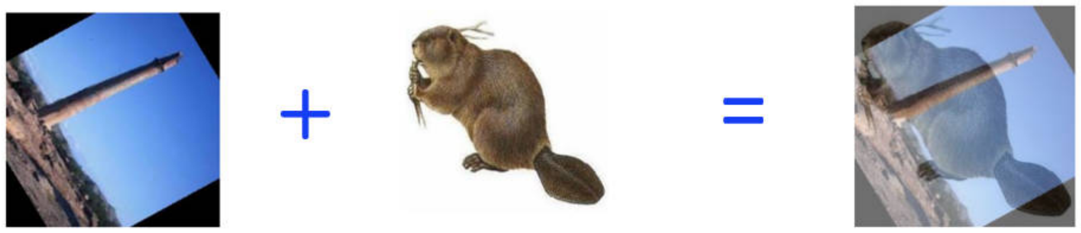


Fig. 4. Example of Mixup

Since the labels are not one-hot encoded, the new images do not have accurate labels. This means that we cannot use accuracy to judge whether overfitting has occurred or whether the model has converged. However, these issues are not critical. It can still be determined whether overfitting has occurred and whether the model has converged by comparing the trends in the loss of the training and validation sets.

3.4 Exponential Moving Average

The exponential Moving Average (EMA) technique was also utilized to reduce loss jitter. Specifically, in each epoch, a new weight is updated according to equation (3).

$$\theta_n = \theta_1 - \sum_{i=1}^{n-1}(1-\alpha^{(n-i)})g_i[19] \tag{3}$$

θ_n represents the model weights at the n_th epoch, θ_1 represents the initial weights, g_i represents the gradient value at epoch i, and α is a weight coefficient that is manually set. Equation (3) assigns higher weight to the most recent data while gradually reducing the weight of past data. This means that it can reduce the impact of individual gradient updates, making parameter updates more stable.

In our implementation, we used the code from Nicolas [19].

3.5 Weight Decay

Weight decay (L2 regularization) was used in the AdamW optimizer. The purpose of L2 regularization is to reduce the weight to a smaller value, which can help to reduce the overfitting problem of the model to some extent.

Briefly, overfitting occurs when the weight w is too large, which makes the model too sensitive to noise and unable to extract the true data distribution. L2 regularization reduces the weight w by adding a penalty term, thereby reducing the model's sensitivity to noise. In practice, regularization is often only applied to weight parameters in convolutional and fully connected layers[20], leaving bias parameters unregularized.

In our implementation of the parameter group, we used the code from WZMIAOMIAO [21].

3.6 Cosine Annealing

The cosine annealing learning rate scheduling strategy[13] was utilized to control the learning rate during the training process. Specifically, the learning rate was decayed in a cosine function manner within each training epoch, from the initial learning rate to a set minimum value. The formula for the learning rate change is shown in Equation (4).

$$(\frac{1+cos(\frac{current\_step\times\pi}{cosine\_steps})}{2}) \times (1-end\_factor)+end\_factor \tag{4}$$

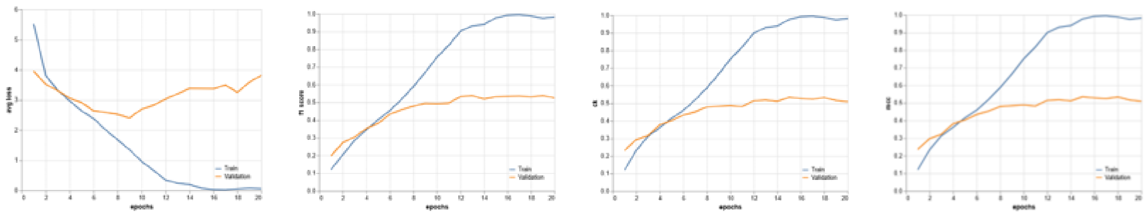
where current step is the current training step, cosine steps is the total number of steps, which equals the number of batches multiplied by the number of epochs. The part $\frac{current\_step\times\pi}{cosine\_steps}$ maps the current training step to the $[0,\pi]$ interval for the cosine function. end factor represents the final value of the learning rate when the training is finished.

At the onset of training, a higher learning rate facilitates rapid convergence for the model. As the training progresses, the gradually diminishing learning rate mitigates loss oscillation and assists the model in locating the optimal minimum.

In our implementation, we used the code from WZMIAOMIAO [22].

4 EXPERIMENT AND RESULTS

Firstly, without using any tricks, the ConvNeXt-tiny model was trained for 20 epochs, and the results are shown in Figure 5.



(a) Average Loss (b) Weighted F1 (c) Cohen's Kappa Score (d) MCC

Fig. 5. Visualization of the Training Process

Based on Figure 5, the training loss continued to decrease, but the validation and test losses stopped improving and started to increase, indicating severe overfitting. Additionally, the Weighted F1[1], MCC[2], and CK[3] on the training set were significantly higher than those on the validation set, confirming the occurrence of overfitting.

To improve the model's generalization ability, Rand augmentation and Mixup were utilized. Additionally, group weight decay was applied to avoid overfitting caused by large weight values, as L2 regularization reduces the value of w by adding a penalty term to it. The model was trained for 30 epochs, and the average loss decreased as shown in Figure 6, with the results presented in Table 1.

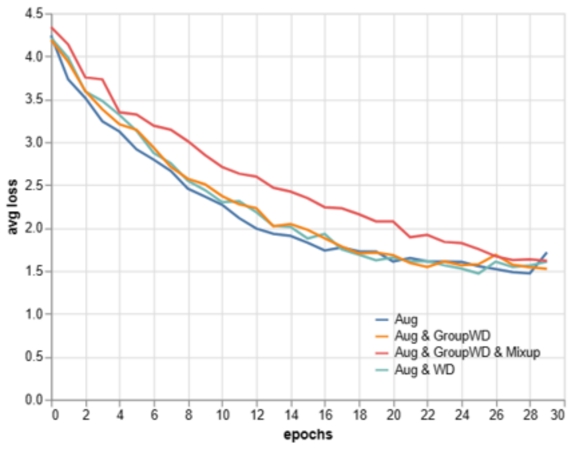


Fig. 6. Average Loss of 4 methods

From figure 6, it can be seen that the average loss of the combined use of data augmentation, group weight decay, and Mixup techniques has been continuously decreasing, while the other methods tend to flatten out or even have an upward trend.

Table 1. The Impact of Different Anti-overfitting Techniques

Techniques	Weighted F1	Cohen's Kappa	MCC
Baseline	0.5256	0.5083	0.5095
Aug	0.6543	0.6471	0.6478
Aug&WD	0.6519	0.6504	0.6512
Aug&GroupWD	**0.6743**	**0.6707**	**0.6714**
Aug&GroupWD&Mixup	0.5892	0.6110	0.6124

From Table 1, we observed that the Aug & GroupWD model had the highest values for all evaluation metrics. However, its loss has almost stabilized and even shows an upward trend, indicating that convergence has nearly completed. On the other hand, the Aug & GroupWD & Mixup model still has a clear downward trend in the average loss, indicating that convergence has not yet been reached. Therefore, the decision was made to continue training for an additional 90 epochs to observe the convergence and determine whether overfitting would occur. The results are presented in Figure 7.

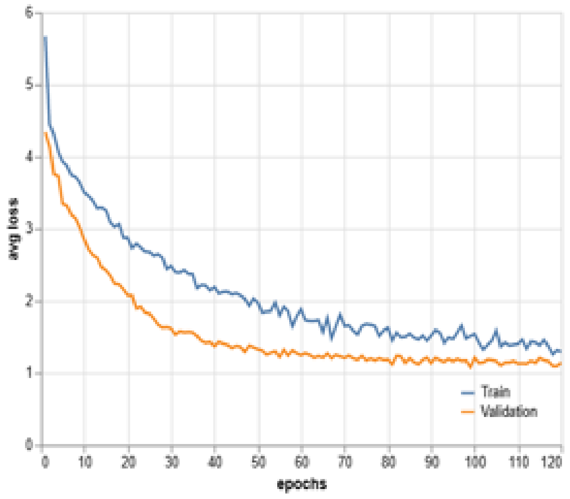


Fig. 7. Avergae Loss of Aug&GroupWD&Mixup on 120 epochs

After 120 epochs, it can be observed from Figure 4 that there was a significant improvement from the previous results.

However, after about 70 epochs, the validation loss stabilizes while the training loss continues to decrease. I think it may not overfitting but rather due to small sample categories may not have received sufficient training. To verify this hypothesis, I print the confusion matrix.

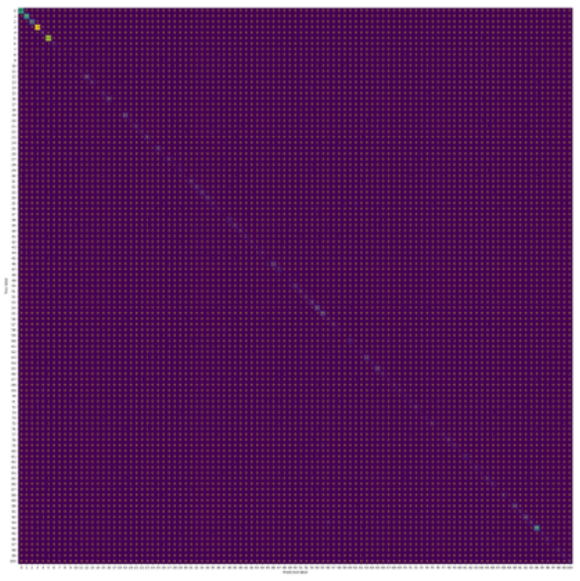


Fig. 8. Confusion Matrix

Due to the large number of categories in the dataset, it was difficult to observe the metric values clearly. As a solution, the dataset was divided into two groups: large-sample categories (those with a quantity greater than or equal to 133) and small-sample categories (those with a quantity less than 133). The average accuracy for both categories are presented below.

Table 2. Average Accuracy for Different Category Sample Size

Classification	Average accuracy
small-sample category (<133)	0.539
large-sample category (≥133)	0.952

From Table 2, the accuracy for small-sample categories is pretty low, which confirms the hypothesis. Therefore, the decision was made to try weighted random sampling and resampling to alleviate this problem, and the model was trained for 60 epochs, respectively.

Table 3. Comparison of Average Accuracy for Different Sampling Methods by Sample Size

Classification	Baseline	Resampling	Weighted Random Sampling
small-sample category (<133)	0.539	**0.582**	**0.582**
large-sample category (≥133)	**0.952**	0.889	0.880

AsAs can be observed from Table 3, the accuracy of weighted random sampling and resampling is the same for small samples. For large samples, a significant decrease in accuracy was observed for both methods, but resampling performed slightly better.

Although resampling has slightly better accuracy, it requires a huge GPU memory. Therefore, the decision was made to use weighted random sampling, and the model was trained for 150 epochs to ensure full convergence.

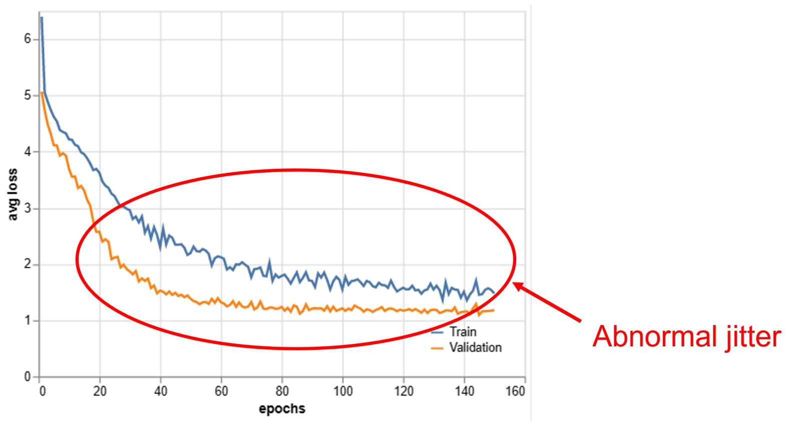


Fig. 9. Average Loss of Weighted Random Sampling

Based on the observation from Figure 9, it was noticed that there was significant jitter and only marginal improvement in accuracy after 30 epochs. To address this issue, three different methods were experimented with: (1) reducing the learning rate by half, (2) increasing the batch size to 96, and (3) implementing the Exponential Moving Average (EMA) on the model weights after completing 60 epochs of training.

Table 4. Comparison of Different Training Strategies

Strategies	Weighted F1	Cohen's Kappa	MCC
Baseline	0.7364	0.7323	0.7327
Half lr	0.7422	0.7534	0.7540
$2 \times$ Batch	0.7396	0.7511	0.7517
EMA	**0.7904**	**0.7931**	**0.7932**

According to Table 4, the EMA method achieves the best results in all metrics, followed by the learning rate reduction method.

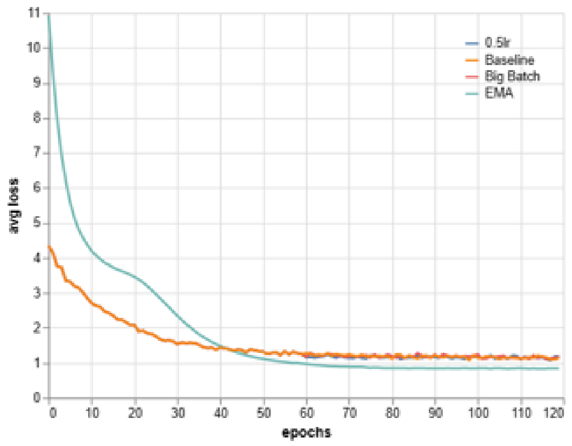


Fig. 10. Average Loss of 3 Training Strategies

From Figure 10, it can be observed that the loss of EMA is very smooth, with almost no jitter. Based on these results, the decision was made to integrate the utilization of learning rate reduction and EMA to further enhance the performance of the model. Specifically, the cosine annealing method was employed to decrease the learning rate, which is currently a popular approach. The initial learning rate was increased to 0.002 and gradually decreased to 1e-6. Due to computational constraints, only two batch sizes were tested. The model was trained for 150 epochs to ensure adequate convergence. The results are presented in Table 5.

Table 5. Comparison of Different Training Strategies

Batch Size	Weighted F1	Cohen's Kappa	MCC	Large-sample	Small-sample
64	**0.8444**	**0.8440**	**0.8442**	**0.962**	**0.649**
8	0.8349	0.8334	0.8335	0.938	0.627

As can be observed from table 5, the larger batch size outperforms the smaller batch size in all metrics.

In summary, weighted random sampling and various data augmentation techniques such as RandAugment and Mixup were employed. The training strategies involved incorporating the EMA technique and cosine annealing with a total of 150 epochs. A final test accuracy of approximately 85% was achieved.

Lastly, to explore the performance limits of ConvNeXt-Tiny, transfer learning was utilized by using the weights pre-trained on ImageNet-1k to obtain the model's upper limit. The model's classification head was replaced with a fully connected layer outputting 101 classes. The average accuracy of transfer learning is displayed in Figure 11.

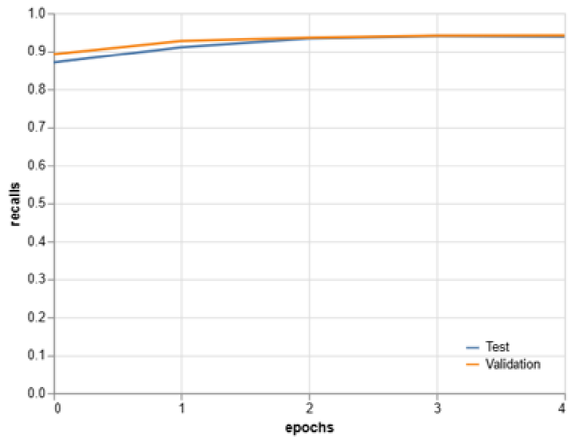


Fig. 11. Average Accuracy of Transfer Learning

From Figure 11, it can be seen that the pre-trained model converges very quickly.

Table 6. Comparison of Average Accuracy for Different Sampling Methods by Sample Size

Classification	Average Accuracy
small-sample category (<133)	0.923
large-sample category (≥133)	0.983

From Table 6, it is evident that the accuracy of small-sample categories is significantly higher than that of our trained model, while the accuracy of large-sample categories is slightly higher than that of our trained model. An accuracy of nearly 95% can be attained in just five epochs.

However, this classification result may not be perfect, as the model was solely trained on the ImageNet-1k dataset. If access to weights trained on the ImageNet-22k dataset were available and the largest ConvNeXt XL model were employed, it is believed that even better performance could be achieved.

REFERENCES

[1] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45:427–437, July 2009. https://doi.org/10.1016/j.ipm.2009.03.002.

[2] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment. *IEEE Access*, 9:78368–78381, May 2021. https://ieeexplore.ieee.org/document/9440903.

[3] Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational Linguistics*, 22, 1996. https://arxiv.org/abs/cmp-lg/9602004.

[4] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022. https://doi.org/10.48550/arXiv.2201.03545.

[5] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *arXiv preprint arXiv:1512.04150*, 2015. https://arxiv.org/abs/1512.04150.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. https://dl.acm.org/doi/10.1145/3065386.

[7] Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, and Stella X Yu. Large-scale long-tailed recognition in an open world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. https://arxiv.org/abs/1904.05160.

[8] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018. https://www.sciencedirect.com/science/article/abs/pii/S0893608018302107.

[9] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 2019. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0.

[10] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018. https://arxiv.org/abs/1710.09412.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

[12] Boris Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[13] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. https://arxiv.org/abs/1608.03983.

[14] 太阳花的小绿豆. ConvNeXt 网络详解. [Online; Accessed: Apr. 22, 2023], Available: https://blog.csdn.net/qq_37541097/article/details/122556545, CSDN blog, 2023.

[15] WZMIAOMIAO. Deep learning for image processing: Convnext implementation. https://github.com/WZMIAOMIAO/deep-learning-for-image-processing/blob/master/pytorch_classification/ConvNeXt/model.py, 2023.

[16] Facebook AI Research. Convnext: A convnet for the 2020s. https://github.com/facebookresearch/ConvNeXt, 2023. Version: main.

[17] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106:59–70, 2007. https://doi.org/10.1016/j.cviu.2005.09.012.

[18] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond empirical risk minimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. https://arxiv.org/abs/1710.09412.

[19] Nicolas. 炼丹技巧-指数移动平均（EMA）的原理及PyTorch实现. [Online; Accessed: Apr. 20, 2023], Available: https://zhuanlan.zhihu.com/p/68748778, ZHIHU blog, 2019.

[20] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *arXiv preprint arXiv:1812.01187*, 2018. https://arxiv.org/abs/1812.01187.

[21] Ziming Wang. Deep learning for image processing. https://github.com/WZMIAOMIAO/deep-learning-for-image-processing, 2022.

[22] WZMIAOMIAO. Deep learning for image processing: Convnext training. https://github.com/WZMIAOMIAO/deep-learning-for-image-processing/blob/master/pytorch_classification/ConvNeXt/train.py, 2022. Commit 01e6b29 on Apr 8, 2022.